

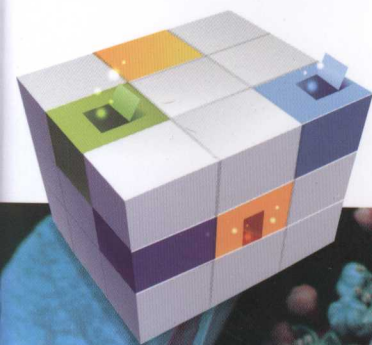
版权注意事项：1、书籍版权归著者和出版社所有；
2、本PDF仅用于个人获取知识，进行私底下知识交流；
3、PDF获得者不得在互联网以任何目的进行传播；
如有需要，请尽量购买正版实体书！支持书籍作者！！



MySQL技术精粹

架构、高级特性、性能优化与集群实战

Architecture, Advanced Features, Performance Optimization and Clustering



张工厂 著

面向MySQL中高级用户，全面、详尽讲解MySQL高级技术
提供大量的数据库使用方法和技巧，让你看得懂、学得会、做得出
将最实用的MySQL技术融入到每个案例中，教你快速成为MySQL数据库顶尖高手



源码，网络存储和集群操作文档

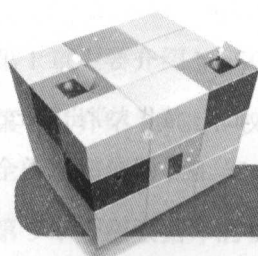


清华大学出版社

内容简介

本书共分10章，第1章介绍MySQL数据库的发展历史、架构、高级特性、性能优化与集群实战。第2章介绍MySQL数据库的安装与配置。第3章介绍MySQL数据库的备份与恢复。第4章介绍MySQL数据库的复制。第5章介绍MySQL数据库的集群。第6章介绍MySQL数据库的优化。第7章介绍MySQL数据库的故障排除。第8章介绍MySQL数据库的安全。第9章介绍MySQL数据库的监控。第10章介绍MySQL数据库的部署。

本书内容



MySQL技术精粹

架构、高级特性、性能优化与集群实战

张工厂 著

清华大学出版社

北京

内 容 简 介

本书针对 MySQL 中高级用户,详细讲解 MySQL 高级使用技术。书中详解了每一个知识点以及数据库操作的方法和技巧。本书注重实战操作,帮助读者循序渐进地掌握 MySQL 中的各项高级技术。

本书主要包括 MySQL 架构介绍、MySQL 权限与安全、MySQL 备份与还原、MySQL 的高级特性、MySQL 锁定机制、使用 MySQL Workbench 管理数据库、SQL 性能优化、MySQL 服务器性能优化、MySQL 性能监控、MySQL Replication、MySQL Cluster 实战、企业中 MySQL 的高可用架构实战。同时,本书还提供了所有示例的源码,读者可以直接查看和调用。

本书适合有一定基础的 MySQL 数据库学习者,MySQL 数据库开发人员和 MySQL 数据库管理人员,同时也能作为高等院校和培训学校相关专业师生的教学参考用书。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

MySQL 技术精粹:架构、高级特性、性能优化与集群实战 / 张工厂著. -- 北京:清华大学出版社,2015
ISBN 978-7-302-42043-9

I. ①M… II. ①张… III. ①关系数据库系统 IV.①TP311.138

中国版本图书馆 CIP 数据核字(2015)第 263505 号

责任编辑:夏非彼

封面设计:王 翔

责任校对:闫秀华

责任印制:宋 林

出版发行:清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址:北京清华大学学研大厦 A 座 邮 编:100084

社总机:010-62770175 邮 购:010-62786544

投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈:010-62772015, zhiliang@tup.tsinghua.edu.cn

印 装 者:北京密云胶印厂

经 销:全国新华书店

开 本:190mm×260mm

印 张:26.25

字 数:672 千字

版 次:2015 年 12 月第 1 版

印 次:2015 年 12 月第 1 次印刷

印 数:1~3000

定 价:79.00 元

产品编号:066559-01

前言

本书是面向 MySQL 数据库管理系统读者的一本高质量的书籍。目前国内 MySQL 需求旺盛，各大知名企业高薪招聘技术能力强的 MySQL 开发人员和管理人员。本书根据这样的需求，针对已经有 MySQL 基础的读者，注重内容实战，通过实例的操作与分析，引领读者快速学习和掌握 MySQL 开发和管理的先进技术。

本书内容

第 1 章主要介绍 MySQL 架构、各种 MySQL 存储引擎的特性。

第 2 章介绍 MySQL 权限与安全。包括权限表、账户管理、权限管理、访问控制、MySQL 的安全问题和使用 SSL 安全连接。

第 3 章介绍数据库的备份还原。包括各种备份方法、各种还原方法、数据库迁移的方法、表的导入和导出。

第 4 章介绍 MySQL 的高级特性。包括 MySQL 查询缓存、合并表和分区表、事务控制和 MySQL 分布式事务。

第 5 章介绍 MySQL 锁定机制。包括 MySQL 锁定机制的概述、MyISAM 表级锁和 InnoDB 行级锁。

第 6 章介绍使用 MySQL Workbench 管理数据库。包括 MySQL Workbench 简介、SQL Development 的基本操作、Data Modeling 的基本操作、Server Administration 的基本操作。

第 7 章介绍 SQL 性能优化。包括优化简介、MySQL Query Optimizer 概述、SQL 语句优化的基本思路、利用 EXPLAIN 分析查询语句、利用 Profiling 分析查询语句、合理地使用索引、不同类型 SQL 语句优化方法、优化数据库结构、分析表、检查表和优化表。

第 8 章介绍 MySQL 服务器性能优化。包括 MySQL 源码安装的性能优化、MySQL 服务器配置优化、MySQL 日志设置优化、MySQL I/O 设置优化、MySQL 并发设置优化、线程、Table Cache 和临时表的优化。

第 9 章介绍 MySQL 性能监控。包括基本监控系统方法、开源监控利器 Nagios 实战、MySQL 监控利器 Cacti 实战。

第 10 章介绍 MySQL Replication。包括 MySQL Replication 概述、Windows 环境下的 MySQL 主从复制、Linux 环境下的 MySQL 复制、查看 Slave 的复制进度、日常管理和维护、切换主从服务器。

第 11 章介绍 MySQL Cluster 实战。包括 MySQL Cluster 概述、Linux 环境下 MySQL Cluster 安装和配置、管理 MySQL Cluster、维护 MySQL Cluster、Windows 操作系统中配置 Cluster。

第 12 章介绍企业中 MySQL 的高可用架构。包括 MySQL 高可用的简单介绍、MySQL 主从复制、MySQL+DRBD+HA、Lvs+Keepalived+MySQL 单点写入主主同步方案、MMM 高可用 MySQL 方案。

本书特色

内容全面，案例丰富：知识点由浅入深，涵盖了所有 MySQL 的实用知识点，由浅入深地掌握 MySQL 数据库管理技术。把知识点融汇于系统的案例实训当中，并且结合综合案例进行讲解和拓展。进而达到“知其然，并知其所以然”的效果。

图文并茂，易学易用：注重操作，图文并茂，在介绍案例的过程中，每一个操作均有对应步骤和过程说明。这种图文结合的方式使读者在学习过程中能够直观、清晰地看到操作的过程以及效果，便于读者更快地理解和掌握。

提示技巧，源码下载：本书对读者在学习过程中可能会遇到的疑难问题以“提示”和“技巧”的形式进行了说明，以免读者在学习的过程中走弯路。为了方便读者学习，本书源码提供下载。

读者对象

本书是一本全面介绍 MySQL 数据库高级技术的教程，内容丰富、条理清晰、实用性强，适合以下读者学习使用：

- 有一定基础的 MySQL 数据库学习者。
- 希望全面、深入掌握 MySQL 的开发人员。
- MySQL 数据库管理员。
- 高等院校和培训学校相关专业的师生。

致谢

参与本书创作的作者除了封面署名人员以外，还有刘玉萍、刘增杰、胡同夫、王英英、

肖品、孙若淞、王攀登、王维维、梁云亮、刘海松、陈伟光、包惠利等人参与了创作。虽然倾注了作者的努力，但由于水平有限、时间仓促，书中难免有错漏之处，请读者谅解，如果遇到问题或有意见和意见，敬请与我们联系，我们将全力提供帮助。

本书技术支持 QQ 群：221376441。

源码下载

本书使用的源码下载地址如下：

<http://pan.baidu.com/s/1ntu60IH>

如果下载有问题，请联系电子邮箱 booksaga@163.com，邮件主题为“MySQL 精粹源码”。

著者

2015 年 11 月

目 录

| | |
|-------------------------------------------|----|
| 第 1 章 MySQL 架构介绍 | 1 |
| 1.1 MySQL 架构 | 1 |
| 1.1.1 MySQL 物理文件的组成 | 2 |
| 1.1.2 MySQL 各逻辑块简介 | 4 |
| 1.1.3 MySQL 各逻辑块协调工作 | 6 |
| 1.2 MySQL 存储引擎概述 | 7 |
| 1.3 MySQL 各种存储引擎的特性 | 10 |
| 1.3.1 MyISAM | 10 |
| 1.3.2 InnoDB | 12 |
| 1.3.3 MEMORY | 15 |
| 1.3.4 MERGE | 18 |
| 1.3.5 BerkeleyDB 存储引擎 | 20 |
| 1.4 MySQL 工具 | 21 |
| 1.4.1 MySQL 命令行实用程序 | 21 |
| 1.4.2 MySQL Workbench | 33 |
| 1.5 本章小结 | 34 |
| 第 2 章 MySQL 权限与安全 | 35 |
| 2.1 权限表 | 35 |
| 2.1.1 user 表 | 35 |
| 2.1.2 db 表和 host 表 | 37 |
| 2.1.3 tables_priv 表和 columns_priv 表 | 39 |
| 2.1.4 procs_priv 表 | 40 |
| 2.2 账户管理 | 41 |
| 2.2.1 登录和退出 MySQL 服务器 | 41 |

| | | |
|-------|------------------------------|----|
| 2.2.2 | 新建普通用户 | 43 |
| 2.2.3 | 删除普通用户 | 47 |
| 2.2.4 | root 用户修改自己的密码 | 48 |
| 2.2.5 | root 用户修改普通用户密码 | 50 |
| 2.2.6 | 普通用户修改密码 | 51 |
| 2.2.7 | root 用户密码丢失的解决办法 | 51 |
| 2.3 | 权限管理 | 53 |
| 2.3.1 | MySQL 的各种权限 | 53 |
| 2.3.2 | 授权 | 55 |
| 2.3.3 | 收回权限 | 57 |
| 2.3.4 | 查看权限 | 58 |
| 2.4 | 访问控制 | 59 |
| 2.4.1 | 连接核实阶段 | 59 |
| 2.4.2 | 请求核实阶段 | 60 |
| 2.5 | MySQL 的安全问题 | 61 |
| 2.5.1 | 操作系统相关的安全问题 | 61 |
| 2.5.2 | 数据库相关的安全问题 | 62 |
| 2.6 | 使用 SSL 安全连接 | 71 |
| 2.7 | 综合管理用户权限 | 77 |
| 2.8 | 小结 | 80 |
| 第 3 章 | 数据备份与还原 | 81 |
| 3.1 | 数据备份 | 81 |
| 3.1.1 | 使用 mysqldump 命令备份 | 81 |
| 3.1.2 | 直接复制整个数据库目录 | 88 |
| 3.1.3 | 使用 mysqlhotcopy 工具快速备份 | 88 |
| 3.2 | 数据还原 | 89 |
| 3.2.1 | 使用 MySQL 命令还原 | 89 |
| 3.2.2 | 直接复制到数据库目录 | 90 |
| 3.2.3 | mysqlhotcopy 快速恢复 | 90 |
| 3.3 | 数据库迁移 | 90 |
| 3.3.1 | 相同版本的 MySQL 数据库之间的迁移 | 91 |
| 3.3.2 | 不同版本的 MySQL 数据库之间的迁移 | 91 |

| | | |
|-------|---------------------------------------|-----|
| 3.3.3 | 不同数据库之间的迁移 | 92 |
| 3.4 | 表的导出和导入 | 92 |
| 3.4.1 | 使用 SELECT...INTO OUTFILE 导出文本文件 | 92 |
| 3.4.2 | 用 mysqldump 命令导出文本文件 | 95 |
| 3.4.3 | 用 MySQL 命令导出文本文件 | 98 |
| 3.4.4 | 使用 LOAD DATA INFILE 方式导入文本文件 | 101 |
| 3.4.5 | 使用 mysqlimport 命令导入文本文件 | 103 |
| 3.5 | 综合实例——数据的备份与恢复 | 105 |
| 3.6 | 小结 | 109 |
| 第 4 章 | MySQL 的高级特性 | 110 |
| 4.1 | MySQL 查询缓存 | 110 |
| 4.1.1 | 认识查询缓存 | 110 |
| 4.1.2 | 监控和维护查询缓存 | 115 |
| 4.1.3 | 如何检查缓存命中率 | 117 |
| 4.1.4 | 优化查询缓存 | 118 |
| 4.2 | 合并表和分区表 | 119 |
| 4.2.1 | 合并表 | 119 |
| 4.2.2 | 分区表 | 121 |
| 4.3 | 事务控制 | 131 |
| 4.4 | MySQL 分布式事务 | 135 |
| 4.4.1 | 了解分布式事务的原理 | 135 |
| 4.4.2 | 分布式事务的语法 | 136 |
| 4.5 | 小结 | 137 |
| 第 5 章 | MySQL 锁定机制 | 138 |
| 5.1 | MySQL 锁定机制概述 | 138 |
| 5.2 | MyISAM 表级锁 | 143 |
| 5.2.1 | MyISAM 表级锁的锁模式 | 143 |
| 5.2.2 | 获取 MyISAM 表级锁的争用情况 | 145 |
| 5.2.3 | MyISAM 表级锁加锁方法 | 146 |
| 5.2.4 | MyISAM Concurrent Insert 的特性 | 148 |
| 5.2.5 | MyISAM 表锁优化建议 | 150 |

| | |
|---------------------------------------------|------------|
| 5.3 InnoDB 行级锁 | 150 |
| 5.3.1 InnoDB 行级锁模式 | 150 |
| 5.3.2 获取 InnoDB 行级锁的争用情况 | 155 |
| 5.3.3 InnoDB 行级锁的实现方法 | 157 |
| 5.3.4 间隙锁 (Net-Key 锁) | 162 |
| 5.3.5 InnoDB 在不同隔离级别下加锁的差异 | 163 |
| 5.3.6 InnoDB 存储引擎中的死锁 | 164 |
| 5.3.7 InnoDB 行级锁优化建议 | 166 |
| 5.4 小结 | 167 |
| 第 6 章 使用 MySQL Workbench 管理数据库 | 168 |
| 6.1 MySQL Workbench 简介 | 168 |
| 6.1.1 MySQL Workbench 的概述 | 168 |
| 6.1.2 MySQL Workbench 的优势 | 169 |
| 6.1.3 MySQL Workbench 的安装 | 169 |
| 6.2 SQL Development 的基本操作 | 171 |
| 6.2.1 创建数据库连接 | 171 |
| 6.2.2 创建新的数据库 | 173 |
| 6.2.3 创建和删除新的数据表 | 174 |
| 6.2.4 添加、修改表记录 | 177 |
| 6.2.5 查询表记录 | 178 |
| 6.2.6 修改表结构 | 178 |
| 6.3 Data Modeling 的基本操作 | 179 |
| 6.3.1 建立 ER 模型 | 179 |
| 6.3.2 导入 ER 模型 | 184 |
| 6.4 Server Administration 的基本操作 | 185 |
| 6.4.1 管理 MySQL 用户 | 186 |
| 6.4.2 备份 MySQL 数据库 | 188 |
| 6.4.3 还原 MySQL 数据库 | 191 |
| 6.5 小结 | 192 |
| 第 7 章 SQL 性能优化 | 193 |
| 7.1 优化简介 | 193 |

- 7.2 MySQL Query Optimizer 概述..... 194
- 7.3 SQL 语句优化的基本思路..... 194
- 7.4 利用 EXPLAIN 分析查询语句..... 196
 - 7.4.1 EXPLAIN 语句的基本语法..... 196
 - 7.4.2 EXPLAIN 语句分析实例..... 208
- 7.5 利用 Profiling 分析查询语句..... 212
- 7.6 合理地使用索引..... 216
 - 7.6.1 索引对查询速度的影响..... 216
 - 7.6.2 如何使用索引查询..... 217
- 7.7 不同类型 SQL 语句优化方法..... 220
 - 7.7.1 优化 INSERT 语句..... 220
 - 7.7.2 优化 ORDER BY 语句..... 221
 - 7.7.3 优化 GROUP BY 语句..... 222
 - 7.7.4 优化嵌套查询..... 223
 - 7.7.5 优化 OR 条件..... 224
 - 7.7.6 优化插入记录的速度..... 226
- 7.8 优化数据库结构..... 228
 - 7.8.1 将字段很多的表分解成多个表..... 228
 - 7.8.2 增加中间表..... 230
 - 7.8.3 增加冗余字段..... 231
- 7.9 分析表、检查表和优化表..... 232
 - 7.9.1 分析表..... 232
 - 7.9.2 检查表..... 233
 - 7.9.3 优化表..... 233
- 7.10 小结..... 234
- 第 8 章 MySQL 服务器性能优化..... 235
 - 8.1 MySQL 源码安装的性能优化..... 235
 - 8.2 MySQL 服务器配置优化..... 238
 - 8.2.1 查看性能参数的方法..... 238
 - 8.2.2 key_buffer_size 的设置..... 243
 - 8.2.3 table_cache 的设置..... 246
 - 8.2.4 内存参数的设置..... 248

| | |
|---------------------------------------|------------|
| 8.2.5 日志和事务参数的设置 | 252 |
| 8.2.6 存储和 I/O 相关参数的设置 | 253 |
| 8.2.7 其他重要参数的设置 | 254 |
| 8.3 MySQL 日志设置优化 | 256 |
| 8.4 MySQL I/O 设置优化 | 257 |
| 8.5 MySQL 并发设置优化 | 259 |
| 8.6 线程、Table Cache 和临时表的优化 | 261 |
| 8.6.1 线程的优化 | 261 |
| 8.6.2 关于 table_cache 相关的优化 | 262 |
| 8.6.3 关于临时表的优化 | 263 |
| 8.7 小结 | 264 |
| 第 9 章 MySQL 性能监控 | 265 |
| 9.1 基本监控系统方法 | 265 |
| 9.1.1 ps 命令 | 265 |
| 9.1.2 top 命令 | 266 |
| 9.1.3 vmstat 命令 | 268 |
| 9.1.4 mytop 命令 | 269 |
| 9.1.5 sysstat 工具 | 272 |
| 9.2 开源监控利器 Nagios 实战 | 277 |
| 9.2.1 安装 Nagios 之前的准备工作 | 277 |
| 9.2.2 安装 Nagios 主程序 | 279 |
| 9.2.3 整合 Nagios 到 Apache 服务 | 280 |
| 9.2.4 安装 Nagios 插件包 | 284 |
| 9.2.5 监控服务器的 CPU、负载、磁盘 I/O 使用情况 | 286 |
| 9.2.6 配置 Nagios 监控 MySQL 服务器 | 291 |
| 9.3 MySQL 监控利器 Cacti 实战 | 293 |
| 9.3.1 Cacti 工具的安装 | 294 |
| 9.3.2 Cacti 监控 MySQL 服务器 | 299 |
| 9.4 小结 | 304 |
| 第 10 章 MySQL Replication | 305 |
| 10.1 MySQL Replication 概述 | 305 |

| | | |
|--------|-------------------------------|-----|
| 10.2 | Windows 环境下的 MySQL 主从复制 | 306 |
| 10.2.1 | 复制前的准备工作 | 306 |
| 10.2.2 | Windows 环境下实现主从复制 | 306 |
| 10.2.3 | Windows 环境下主从复制测试 | 314 |
| 10.3 | Linux 环境下的 MySQL 复制 | 315 |
| 10.3.1 | 下载并安装 MySQL 5.6 | 315 |
| 10.3.2 | 单机主从复制前的准备工作 | 316 |
| 10.3.3 | mysqld_multi 实现单机主从复制 | 320 |
| 10.3.4 | 不同服务器之间实现主从复制 | 328 |
| 10.3.5 | MySQL 主要复制启动选项 | 329 |
| 10.3.6 | 指定复制的数据库或者表 | 330 |
| 10.4 | 查看 Slave 的复制进度 | 338 |
| 10.5 | 日常管理和维护 | 339 |
| 10.5.1 | 了解服务器的状态 | 339 |
| 10.5.2 | 服务器复制出错的原因 | 340 |
| 10.6 | 切换主从服务器 | 343 |
| 10.7 | 小结 | 347 |
| 第 11 章 | MySQL Cluster 实战 | 348 |
| 11.1 | MySQL Cluster 概述 | 348 |
| 11.1.1 | MySQL Cluster 基本概念 | 348 |
| 11.1.2 | 理解 MySQL Cluster 节点 | 349 |
| 11.2 | Linux 环境下 MySQL Cluster 安装和配置 | 350 |
| 11.2.1 | 安装 MySQL Cluster 7.2.8 软件 | 352 |
| 11.2.2 | 管理节点配置步骤 | 357 |
| 11.2.3 | 配置 SQL 节点和数据节点 | 358 |
| 11.3 | 管理 MySQL Cluster | 358 |
| 11.3.1 | Cluster 的启动 | 358 |
| 11.3.2 | Cluster 的测试 | 360 |
| 11.3.3 | Cluster 的关闭 | 363 |
| 11.4 | 维护 MySQL Cluster | 363 |
| 11.4.1 | Cluster 的日志的管理 | 366 |
| 11.4.2 | Cluster 的联机备份 | 367 |

| | |
|--------------------------------------------|------------|
| 11.4.3 Cluster 的数据恢复..... | 368 |
| 11.5 Windows 操作系统中配置 Cluster | 369 |
| 11.6 小结 | 374 |
| 第 12 章 企业中 MySQL 的高可用架构 | 375 |
| 12.1 MySQL 高可用的简单介绍 | 375 |
| 12.2 MySQL 主从复制 | 375 |
| 12.2.1 MySQL 主从架构设计 | 376 |
| 12.2.2 配置环境 | 376 |
| 12.2.3 服务器的安装配置 | 376 |
| 12.2.4 LVS 的安装配置 | 379 |
| 12.3 MySQL+DRBD+HA | 381 |
| 12.3.1 什么是 DRBD | 381 |
| 12.3.2 MySQL+DRBD+HA 架构设计 | 382 |
| 12.3.3 配置环境 | 382 |
| 12.3.4 安装配置 Heartbeat | 383 |
| 12.3.5 安装配置 DRBD | 385 |
| 12.4 Lvs+Keepalived+MySQL 单点写入主主同步方案 | 388 |
| 12.4.1 配置环境 | 388 |
| 12.4.2 Lvs+Keepalived 的安装 | 393 |
| 12.4.3 Lvs+Keepalived 的配置 | 394 |
| 12.4.4 Master 和 Backup 的启动 | 397 |
| 12.5 MMM 高可用 MySQL 方案 | 397 |
| 12.5.1 MMM 的架构 | 398 |
| 12.5.2 配置环境 | 398 |
| 12.5.3 MMM 的安装 | 402 |
| 12.5.4 Monitor 服务器的配置 | 402 |
| 12.5.5 各个数据库服务器的配置 | 404 |
| 12.5.6 MMM 的管理 | 404 |
| 12.6 小结 | 405 |

第 1 章

◀ MySQL 架构介绍 ▶

开源数据库 MySQL 功能日益完善，备受企业喜欢。本章主要从 MySQL 的逻辑组成、数据库存储数据引擎，以及 MySQL 相关工具方面介绍 MySQL 的整体架构，让读者能从整体上把握 MySQL，理解 MySQL 的各个逻辑层次是如何协同工作的。

1.1 MySQL 架构

MySQL 服务器由 SQL 层和存储引擎层构成。SQL 层主要功能包括权限判断、SQL 解析功能和查询缓存处理等，存储引擎层（Storage Engine Layer）完成底层数据库数据存储操作。MySQL 整体架构的 SQL 层和存储引擎层实际上各自都包含了很多的小模块，各个模块的工作方式如图 1-1 所示。

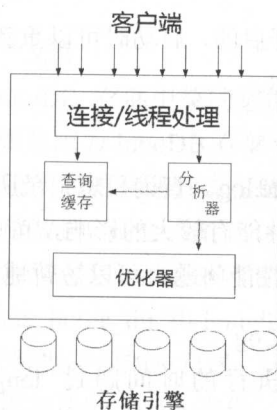


图 1-1 MySQL 各个模块的工作方式

从上图可以看出 MySQL 各个模块的特点如下：

(1) 客户端通过连接/线程处理层来连接 MySQL 数据库，连接/线程处理层主要用来处理客户端的请求、身份验证和数据库安全性验证等。

(2) 查询缓存和查询分析器是 SQL 层的核心部分，其中主要涉及查询的解析、优化、缓

存，以及所有内置的函数，存储过程，触发器，视图等功能。

(3) 优化器主要负责存储和获取所有存储在 MySQL 中的数据。

可以把这三层统称为 MySQL 数据库的 SQL 层。

1.1.1 MySQL 物理文件的组成

MySQL 的物理文件包括日志文件、数据文件和其他文件，下面将详细介绍这些文件的含义和作用。

1. 日志文件

在 MySQL 数据库中，日志文件主要记录了数据库操作信息和错误信息。常用的日志文件包括错误日志、二进制日志、查询日志、慢查询日志和 InnoDB 引擎在线 Redo 日志等。

(1) 错误日志：Error Log

错误日志文件记录了 MySQL Server 运行过程中遇到的所有严重的错误信息，以及 MySQL 每次启动和关闭的详细信息。默认情况下错误日志功能是关闭的，启动时要重新配置 `--log-error[=file_name]` 选项，修改错误日志存放的目录和文件名称。

(2) 二进制日志：Binary Log

二进制日志文件就是常说的 binlog。二进制日志记录了 MySQL 所有修改数据库的操作，然后以二进制的形式记录在日志文件中，其中还包括每条语句所执行的时间和所消耗的资源，以及相关的事务信息。

默认情况下二进制日志功能是开启的，启动时可以重新配置 `--log-bin[=file_name]` 选项，修改二进制日志存放的目录和文件名称。

(3) 查询日志：Query Log

默认的查询日志文件是 `hostname.log`。查询日志记录所有的查询操作，包括所有的 `select` 操作信息，体积比较大，开启后对性能有较大的影响，可以通过 `--log[=file_name]` 选项开启。如果需要跟踪某些特殊的 SQL 性能问题，可以短暂地打开该功能。

(4) 慢查询日志：Slow Query Log

慢查询日志是指所有 SQL 执行的时间超过 `long_query_time` 变量的语句和达到 `min_examined_row_limit` 条距离的语句。用户可以针对这部分语句性能调优。慢查询日志通过设置 `--log-slow_queries[=file_name]` 选项开启后，将记录日志所在的路径和名称。MySQL 系统默认的慢查询日志的文件名是 `hostname-slow.log`，默认目录也是 `data` 目录。查看慢查询日志可以采用 `mysqldumpslow` 命令对慢查询日志进行分析。

(5) InnoDB 引擎在线 Redo 日志：InnoDB redo Log

InnoDB 引擎在线 Redo 日志记录了 InnoDB 所做的所有物理变更和事务信息。通过 Redo

日志和 Undo 信息, InnoDB 大大地加强了事务的安全性。InnoDB 在线 Redo 日志默认存放在 data 目录下面, 可以通过设置 `innodb_log_group_home_dir` 选项来更改日志的存放位置, 通过 `innodb_log_files_in_group` 选项来设置日志的数量。

2. 数据文件

MySQL 数据库会在 data 目录下面建立一个以数据库为名字的文件夹, 用来存储数据库中的表文件数据。不同的数据库引擎, 每个表的扩展名也不一样, 例如, MyISAM 引擎用“.MYD”作为扩展名, InnoDB 引擎可以用“.ibd”作为扩展名, CSV 引擎使用“.csv”扩展名。

(1) “.frm”文件

无论是哪种存储引擎, 创建表之后就一定会生成一个以表名命名的“.frm”文件。frm 文件主要存放与表相关的数据信息, 主要包括表结构的定义信息。当数据库崩溃时, 用户可以通过 frm 文件来恢复数据表结构。

(2) “.MYD”文件

MyISAM 存储引擎创建表时, 每一个 MyISAM 类型的表都会有一个“.MYD”文件与之对应。“MYD”文件主要用来存放数据表的数据文件。

(3) “.MYI”文件

每一个 MyISAM 类型的表都会有一个“.MYD”文件和一个“.MYI”文件, 对于 MyISAM 存储引擎来说, 可以被缓存的内容主要就是源于“.MYI”文件中, “.MYI”文件中主要用来存储表数据文件中任何索引的数据树。

(4) “.ibd”文件和“.ibdata”文件

这两种文件主要是用来存储 InnoDB 存储引擎的数据, 其中主要包括索引信息。InnoDB 存储引擎采用这两种数据文件, 主要是因为 InnoDB 存储引擎的存储方式能够通过配置来决定是采用共享表空间, 还是采用独享表空间的存储方式存储数据。

如果采用共享表空间的方式存储数据, 则会采用 ibdata 文件来存储, 所有的表共同使用一个或者多个 ibdata 文件。如果采用独享表空间的方式存储数据, 则会采用 ibd 文件来存储。

共享表空间存储通过 `innodb_data_home_dir` 和 `innodb_data_file_path` 两个参数共同配置组成, `innodb_data_home_dir` 参数配置数据存放的总目录, `innodb_data_file_path` 参数配置每一个文件的路径及文件名称。如果需要添加新的 ibdata 文件, 则需要在 `innodb_data_file_path` 参数后面配置, 然后重新启动服务器才能够生效。

3. 其他文件

MySQL 数据库系统除了日志文件、数据文件外, 还包括其他的一些文件。例如系统配置文件、pid 文件、socket 文件等等。

MySQL 系统配置文件一般都在“etc/my.cnf”中。pid 文件类似于 Unix/Linux 操作系统下

面的进程文件，MySQL 服务器的 pid 文件用来存放自己的进程 ID。MySQL 服务器启动后，socket 文件自动生成，该文件主要用来连接客户端。

1.1.2 MySQL 各逻辑块简介

MySQL 逻辑架构采用 SQL 层和存储引擎分离的方式，真正实现了数据存储和逻辑业务的分离，MySQL 的 SQL 层从宏观上可以分为三层，事实上 SQL 层包含了很多的子模块。下面就详细介绍 SQL 层各个子模块的功能。

1. 初始化模块

初始化模块就是在数据库启动的时候，对整个数据库做的一些初始化操作，例如，各种系统环境变量的初始化，各种缓存、存储引擎初始化设置等。

在 MySQL 初始化过程中，部分系统参数是通过 MySQL 数据库系统文件设置的。MySQL 系统参数可以通过“mysqld --verbose -help”命令来查看当前系统所有参数的设置。Linux 平台上 MySQL 数据库读取文件首先会读取/etc/my.cnf 文件，该选项主要是用来设置 MySQL 全局选项，许多初学者在 Linux 平台上安装 MySQL 失败就是因为/etc/my.cnf 的设置是系统默认的错误路径，对于初学者，可以将\$MySQL_HOME/support_files/目录下面的配置文件复制到/etc/my.cnf 中，命令如下：

```
cp ./support_files/my_medium.cnf /etc/my.cnf
```

MySQL 数据库读取完/etc/my.cnf 之后，接下来会解析\$MySQL_HOME/my.cnf。在这个过程中，服务器会到 MySQL 安装目录下面解析数据库的相关配置。MySQL 启动初始化接着会解析 defaults-extra-file 附带选项，修改该参数可以指定系统配置文件，接下来数据库会解析有关用户的选项。

2. 核心 API

MySQL 数据库核心 API 主要实现了数据库底层操作的优化功能，其中主要包括 IO 操作、格式化输出、高性能存储数据结果算法的优化，字符串的处理，其中最重要的是内存管理。

3. 网络交互模块

MySQL 底层相互交互的模块抽象出接口，对外提供可以接收和发送数据的 API 接口，其他模块需要交互的时候，可以通过 API 接口调用。

4. 服务器客户端交互协议模块

MySQL 服务器采用 C/S 的形式访问数据库，数据连接使用 MySQL C/S 交互协议模块，实现了客户端与服务器端交互过程中所需要的一些独特的协议，这些协议都是建立在现有的网络协议之上。

5. 用户模块

用户模块主要功能是用于控制用户登录连接的权限和用户的授权管理。

6. 访问控制模块

访问控制模块主要用于监控用户的每一个操作。访问控制模块实现的功能就是根据用户模块中不同的用户授权,以及根据其数据库的各种约束来控制用户对数据的访问。用户模块和访问控制模块结合起来,就组成了 MySQL 数据库的权限管理功能。

7. 连接管理、连接线程和线程管理

连接管理模块负责监听 MySQL Server 的各种请求,根据不同的请求,然后转发到线程管理模块,每个客户请求都会被数据库自动分配一个独立的线程为其单独服务,而连接线程的主要工作就是负责 MySQL Server 与客户端通信,线程管理模块负责管理这些生成的线程。

8. 转发模块

客户端连接 MySQL 之后会发送一些查询语句,在 MySQL Server 里面,连接线程接收到客户端的一个请求后,会直接将查询转发到各个对应的处理模块。转发模块主要就是根据查询语句语法分析,然后转发给不同的模块处理。

9. 缓存模块

查询缓存模块主要功能是将客户端查询的请求返回的结果集到缓存中,与查询的一个 HASH 值对应。在查询的基表发生任何数据变化后,MySQL 会自动将其查询的缓存失效。

在读写比例非常高的应用系统中,查询缓存对性能的提高是非常显著的。

10. 优化器模块

这个模块主要是将客户端发送的查询请求,在之前算法的基础上分析,计算出一个最优的查询策略,优化之后会提高查询访问的速度,最后根据其最优策略返回查询语句。

11. 表变更管理模块

表变更管理模块主要负责完成 DML 和 DDL 的查询,例如,insert, update, delete, create table, alter table 等语句的处理。

12. 表维护模块

表维护模块主要用于检测表的状态,分析、优化表结构,以及修复表。

13. 系统状态管理模块

在客户端请求系统状态的时候,系统状态模块主要负责将各种状态的数据返回给用户。最常用的一些查询状态的命令包括 show status, show variables 等,都是通过这个模块负责返回的。

14. 表管理器

表管理器主要就是维护系统生成的表文件。例如,MyISAM 存储引擎类型表生成的是 frm 文件、MYD 文件以及 MYI 文件,表管理器的工作就是维护这些文件,将各个表结构的信息缓存起来,另外该模块还管理表级别的锁。

15. 日志记录模块

日志记录模块主要负责整个数据库逻辑层的日志文件，其中包含错误日志，二进制日志，以及慢查询日志等。

16. 复制模块

复制模块分为 Master 模块和 Slave 模块两部分。Master 模块主要负责复制环境中读取 Master 端的 binary 日志，以及 Slave 端的 I/O 线程交互等工作。Slave 模块主要有两个线程，一个负责从 Master 请求和接收 binary 日志，并写入本地 I/O 线程；另一个从 relay log 读取日志事件，然后解析成可以在 Slave 端执行的命令，然后交给 Slave 端的 SQL 线程。

17. 存储引擎接口模块

MySQL 实现了其数据库底层存储引擎的插件式管理，将各种数据处理高度抽象化。

1.1.3 MySQL 各逻辑块协调工作

MySQL 各逻辑块协调工作的过程如图 1-2 所示。

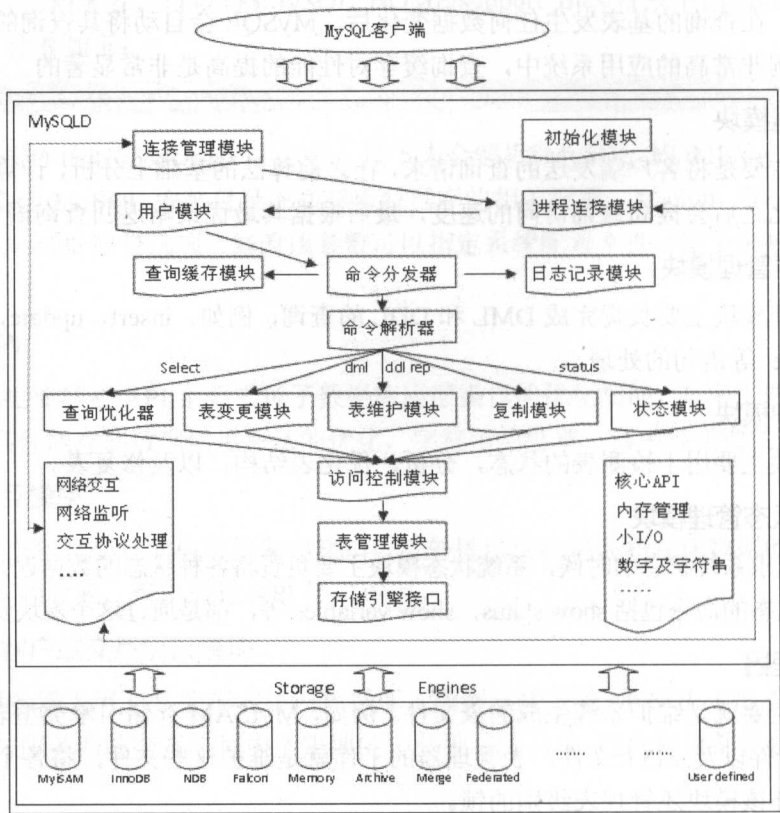


图 1-2 MySQL 各逻辑块协调工作图

MySQL 启动后，MySQL 的初始化模块就从系统配置文件中读取系统参数和命令参数，并

按照参数初始化整个系统，同时存储引擎也会启动，当初始化工作结束后，连接管理模块会监听并接收客户端的程序，连接管理模块会将连接请求转发给线程管理模块去请求一个连接线程。

线程模块接到请求之后会调用用户模块进行授权检查，通过检查后，会检测线程池里是否有空闲连接线程。如果有，就取出跟客户端连接上；如果没有，则建立一个新的线程与客户端建立连接。

MySQL 数据库中的请求有两种，一种是需要命令解析和分发模块解析才能执行请求的操作，另一种是不需要转发就可以直接执行的命令。此时，如果初始阶段开启日志功能，这时候日志模块将请求记入日志，不管是哪种请求，都会记录到日志。

Query 类型的请求，会将控制权交给 Query 解析器，Query 解析器会检查是否是 select 类型的查询；如果是，则启动查询缓存模块，此时会查询缓存中已经存在结果集；如果存在，则将缓存中的数据返回给连接线程模块，之后连接线程会将数据传递到客户端；如果没有被缓存，或者不是一个可以缓存的查询，此时查询解析器进行相应的处理，通过查询分发器给相关的处理模块。

如果解析器结果是 DML/DDl，则交给变更管理模块；如果是一些检查，修复类的查询，则交给表维护模块去处理；如果是一条没有被缓存的查询语句，则交给查询优化器模块。实际上表变更管理器又分为若干小的模块，例如：insert 处理器、delete 处理器、update 处理器、create 处理器，以及 alter 处理器这些小模块来负责不同的 DML 和 DDL。总之，查询优化器，表变更模块，表维护模块，复制模块，状态模块都是根据命令解析器的结果不同而分发给不同类型的模块。

当一条命令执行完成之后，控制权都会还给连接线程模块，在上面各个模块处理过程中，各个模块都依赖于整个 MySQL 的核心 API 模块，比如内存管理，文件 I/O，字符串处理等。

1.2 MySQL 存储引擎概述

MySQL 采用插件的方式，将存储引擎直接加载到正在运行的 MySQL 中，这是 MySQL 数据库的一个重要特性。MySQL5.0 以后的版本支持的数据引擎包括 MyISAM、InnoDB、BDB、MEMORY、EXAMPLE、CSV、NDB Cluster、BLACKHOLE、FEDERATED、Archive 等，如表 1-1 中分析了常用存储引擎的对比情况。

表 1-1 常用存储引擎的对比情况

| 特点 | InnoDB | MyISAM | MEMORY | MERGE | NDB |
|-------|--------|--------|--------|-------|-----|
| 存储限制 | 64TB | 有 | 有 | 没有 | 有 |
| 事务安全 | 支持 | | | | |
| 锁机制 | 行锁 | 表锁 | 表锁 | 表锁 | 行锁 |
| B 数索引 | 支持 | 支持 | 支持 | 支持 | 支持 |

(续表)

| 特点 | InnoDB | MyISAM | MEMORY | MERGE | NDB |
|--------|--------|--------|--------|-------|-----|
| 哈希索引 | | | 支持 | | 支持 |
| 全文索引 | | 支持 | | | |
| 集群索引 | 支持 | | | | |
| 数据缓存 | 支持 | | 支持 | | 支持 |
| 索引缓存 | 支持 | 支持 | 支持 | 支持 | 支持 |
| 数据可压缩 | | 支持 | | | |
| 空间使用 | 高 | 低 | N/A | 低 | 低 |
| 内存使用 | 高 | 低 | 中等 | 低 | 高 |
| 批量插入速度 | 低 | 高 | 高 | 高 | 高 |
| 支持外键 | 支持 | | | | |

查看当前默认的存储引擎，可以使用下面的命令：

```
mysql> show variables like 'table_type';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| table_type    | MyISAM |
+-----+-----+
1 row in set (0.08 sec)
```

可以通过下面的方法查询当前 MySQL 支持的存储引擎。

```
mysql> show engines \g ;
Engine:MyISAM
Support:DEFAULT
Comment:Default engine as of MySQL 5.6 with great performance

Engine:MEMORY
Support:DEFAULT
Comment:Default engine as of MySQL 5.6 with great performance

Engine:InnoDB
Support:YES
Comment:Supports transactions, row-level locking, and foreign keys

Engine:InnoDB
Support:YES
Comment:Supports transactions, row-level locking, and foreign keys

Engine:BerkeleyDB
Support:NO
Comment:Supports transactions and page-level locking
```

```

Engine:BLACKHOLE
Support:YES
Comment:/dev/null storage engine (anything you write to it disappears)

Engine:EXAMPLE
Support:NO
Comment:Example storage engine

Engine:ARCHIVE
Support:YES
Comment:Archive storage engine

Engine:CSV
Support:YES
Comment:CSV storage engine

Engine:ndbcluster
Support:DISABLED
Comment:Clustered, fault-tolerant, memory-based tables

Engine:FEDERATED
Support:DISABLED
Comment:Federated MySQL storage engine

Engine:MRG_MYISAM
Support:YES
Comment:Collection of identical MyISAM tables

Engine:ISAM
Support:NO
Comment:Obsolete storage engine

```

查询当前 MySQL 数据库支持的存储引擎还可以使用如下命令：

```
mysql> show variables like 'have%';
22 rows in set (0.00 sec)
```

MySQL 数据库在创建表的时候，可以添加默认的存储引擎，例如下面的例子：

```
create table books(
  id integer primary key auto_increment,
  name varchar(20) not null
)engine=MyISAM default charset=gbk;
```

也可以使用 `alter table` 命名修改 `book` 表，将其存储引擎修改成其他的存储引擎，如下所示：

```
mysql> alter table books engine = innodb;
```



```
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> show create table books;
mysql> show create table books;
CREATE TABLE `books` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(20) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=gbk
1 row in set (0.01 sec)
```

1.3 MySQL 各种存储引擎的特性

尽管 MySQL 数据库支持种类繁多的数据存储引擎，但是数据存储不管是使用哪种存储引擎，所有的存储数据都被记录到 .frm 文件中，该文件记录了存储的数据，以及表的一些属性值。值得注意的是，不管是使用哪种数据存储引擎都使用了高速缓存，数据库读取 .frm 文件信息后会将表的信息缓存起来，提高了服务器下次读取数据的速度。

1.3.1 MyISAM

在 Linux 平台下，MySQL 数据库默认的数据存储引擎是 MyISAM。MyISAM 存储引擎在 MySQL 数据引擎中起源最早。MyISAM 的最初版本是 ISAM，最初的 MySQL 架构比较混乱，数据业务和数据存储没有很好地分离出来，当出现可以尝试扩展 MySQL 存储引擎的时候，ISAM 被重构，此时 MySQL 从架构上来讲真正实现了逻辑业务和数据存储的分离。

1. MyISAM 文件格式

MyISAM 在磁盘上存储三个文件，文件名和对应的表名是一致的。

- frm 文件：存储表的定义数据。
- MYD 文件：存放表具体记录数据。
- MYI 文件：存储索引。

.frm 文件和 .MYI 文件可以放在不同的目录下面，平均分配读取权限，可以获得更快的速度。MyISAM 存储引擎不支持事务，也不支持主键，对数据的存储和批量查询的速度比较快。

在实际应用中，往往对于不需要完整的事务，主要以查询和增加记录为主的应用采用 MyISAM 存储引擎。MyISAM 存储引擎只缓存索引，对数据文件采用操作系统缓存，如果索引数据超过系统所分配的缓存空间时也会采用操作系统来缓存索引。

2. MyISAM 文件修复

MyISAM 类型的表在数据存储的过程中可能会发生错误,此时可以通过使用 CHECK TABLE 语句检测 MyISAM 表的状态,然后用 REPAIR TABLE 语句修复损坏的表。

另外,还可以使用 myisamchk 工具修复损坏的表。通过 myisamchk 可以检测 MyISAM 类型表的健康状态和修复甚至优化表的存储。在 Linux 平台下,使用 myisamchk 检测某个表的时候,如果有用户同时也在操作这张表,即便这张表是没有问题的,也很有可能会提示该表已经被损坏,所以在检测数据库表的时候,应当停止 mysqld 服务。如果不想停掉 mysqld 服务,至少应该做一个 mysqladmin flush-tables 操作。

使用 myisamchk --help 命令可以获取 myisamchk 的用法, myisamchk 的命令格式如下:

```
myisamchk [options] tables[.MYI]
```

其中[options]用于指定 myisamchk 的操作,具体可选的参数含义如下:

- -c: 表示用来检测表可能存在的错误。
- -d: 表示打印出表的一些信息。
- -e: 表示彻底地检查表。
- -i: 表示打印有关被检查表的信息统计。
- -k #: 一般与参数-r一起使用,表示仅仅更新头#个索引,较高编号的索引被撤销。撤销的索引会通过 myisamchk -r 被重新激活。
- -q: 与-r一起可以使得修复速度更快。
- -r: 表示可以修复几乎所有的损坏的问题。
- -o: 相比-r而言, -o 恢复表的能力要慢些,但是能处理一些-r不能处理的情况。
- -u: 解开一个用 myisampack 压缩的表。
- -w: 表示如果表被锁定,将会发生等待。

例如以下案例演示操作:

```
[root@localhost books]# ls
books.frm books.MYD books.MYI db.opt
[root@localhost books]# myisamchk -e books.MYI
Checking MyISAM file: books.MYI
Data records:      0   Deleted blocks:      0
- check file-size
- check record delete-chain
- check key delete-chain
- check index reference
- check data record references index: 1
- check record links
```

其中-O var=option 这个参数可以用来设置一些变量, -O 选项可能的变量如下:

```
key_buffer_size      20971520
```

```
key_cache_block_size      1024
myisam_block_size         1024
read_buffer_size          2097152
write_buffer_size         2097152
sort_buffer_size          20971520
```

当在很大文件上运行 `myisamchk` 的时候，往往需要较大的内存，默认使用 3MB 来修复。可以通过如下命令来设置内存空间。

```
[root@localhost books]# myisamchk -O sort_key_blocks=16 -e books.MYI
Checking MyISAM file: books.MYI
Data records:      0   Deleted blocks:      0
- check file-size
- check record delete-chain
- check key delete-chain
- check index reference
- check data record references index: 1
- check records and index references
```

3. MyISAM 表的存储格式

MyISAM 类型的表支持三种不同存储类型格式的表，分别如下：

- 静态（固定长度）表。
- 动态可变长度表。
- 压缩表。

其中静态和动态表数据存储根据其表中数据列的类型自动选择，静态表是默认的存储格式，压缩表则只能通过 `myisampack` 工具创建。

静态表中的字段都是固定非变长的字段，这样每个记录都是固定长度的，这种存储方式的优势在于存储速度非常快，容易缓存，表发生缺损后容易恢复，缺点是静态表所占用的空间往往要比动态表多。

MyISAM 存储引擎采用动态表将会支持动态可变长度，字符型的列长是可变的，除了小于 4 个字符的。动态可变长字符通常比静态固定格式需要更少的存储空间，由于采用动态可变长度存储，所以出错的时候也比静态格式恢复更加困难，因为行变化如果很大的话，会被分成碎片。这个时候可以使用 `myisamchk -ei` 获取表的统计信息，并使用 `myisamchk -r` 来进行修复。

MyISAM 存储压缩表需要的磁盘存储空间最小，数据库系统提供了 `myisampack` 工具可以用来压缩 MyISAM 表，每行单独压缩，每列的压缩也不一样。

1.3.2 InnoDB

InnoDB 存储引擎是第三方公司开发的，目前应用最广泛的数据存储引擎除了 MyISAM 之

外就是 InnoDB 了, InnoDB 写的处理相对于 MyISAM 效率低一些, InnoDB 牺牲了存储和查询的效率, 支持事务安全, 支持自动增长列, 对事务安全的支持, 这是 InnoDB 成为 MySQL 最为流行的存储引擎之一的重要原因。下面重点介绍 InnoDB 存储引擎的特点。

1. 支持事务

MySQL 支持对 InnoDB 存储事务控制, 实现了 SQL92 标准所定义的 4 个级别(read uncommitted, repeatable read, read committed 和 serializable)。MySQL 通过 commit、rollback、set autocommit、start transaction 等语法支持本地事务, 具体语法如下所示:

```
START TRANSACTION | BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET AUTOCOMMIT = {0|1}
```

2. 自动增长列

InnoDB 表的自动增长需要在列的后面添加 auto_increment 属性, 表在添加数据的过程中, 可以插入空值, 该列可以自动增加数据。

例如以下实例, 可以实现自动增长列。

首先创建具有增长列属性的表, 命令如下:

```
mysql> create table authors(
-> id integer primary key auto_increment,
-> name varchar(10)
-> )engine=innodb default charset=gbk;
Query OK, 0 rows affected (0.25 sec)
```

插入 id 列为空值的数据, 命令如下:

```
mysql> insert into authors(name) values('ivan'),('susan'),('shark');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

查询插入的结果, 命令如下:

```
mysql> select * from authors;
+----+-----+
| id | name |
+----+-----+
| 1  | ivan |
| 2  | susan |
| 3  | shark |
+----+-----+
3 rows in set (0.06 sec)
```

从结果可以看出, id 列自动插入了数值。

3. 外键约束

InnoDB 实现了外键这一数据库重要功能，从数据库性能上讲数据库外键降低了数据库查询的效率，数据库表之间的耦合度更加紧密，但是对于不少用户来讲，采用外键约束可能是最低成本的选择方式。MySQL 支持外键的存储引擎只有 InnoDB。在创建外键的时候，要求父表必须有对应的索引，子表在创建外键的时候会添加相对应的索引。

下面是一个外键的例子，sclass 是主表，id 作为主表主键索引，st 表示子表，其中 class_id 作为外键对应 sclass 表的 id 值。

首先创建数据表 sclass，命令如下：

```
mysql> create table sclass(
-> id integer primary key auto_increment,
-> cname varchar(20) not null,
-> last_update timestamp not null default current_timestamp on update
current_timestamp)
-> engine=innodb default charset=gbk;
Query OK, 0 rows affected (0.09 sec)
```

创建数据表 st，并添加外键约束，命令如下：

```
mysql> create table st(
-> id integer primary key auto_increment,
-> sname varchar(20) not null,
-> class_id integer not null,
-> last_update timestamp not null default current_timestamp on update
current_timestamp,
-> foreign key(class_id) references sclass(id) on delete restrict on update
cascade
-> )engine=innodb default charset=gbk;
Query OK, 0 rows affected (0.02 sec)
```

对于上面创建的两个表，在删除操作时，如果是删除主表的数据，子表对应的记录不会被删除；如果是更新主表，子表对应的记录会更新。

在物理存储方面，InnoDB 有自己独特的存储方式，数据也是存放在 .frm 文件里面，但是表数据和索引数据是存放在一起的。InnoDB 的存储表和索引有以下两种方式：

(1) 使用共享表空间存储，也就是所有表和索引数据存放在同一个表空间中，数据和索引在 innodb_data_home_dir 和 innodb_data_file_path 定义的表空间中，可以使用一个或者多个数据文件。

(2) 使用多表空间存储，这种存储方式创建的表结构存放在 .frm 文件中，但是每个表的数据和索引被存放在一个单独的 .ibd 文件中。如果是分区表，则每个分区对应单独的 .ibd 文件，文件名称是“表名+分区名”，可以在创建分区的时候指定每个分区的数据文件的路径，这样的好处在于可以将表的读取操作平均分布到若干个磁盘分区文件上，提高数据访问的效率。

要使用多表空间存储方式，需要设置 `innodb_file_per_table` 参数。该参数可以修改 InnoDB 为独立表空间模式，每个数据库的表都会生成一个数据空间。

可以使用如下方式查看多表存储空间模式是否已经开启：

```
mysql> show variables like '%per_table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| innodb_file_per_table | OFF |
+-----+-----+
1 row in set (0.00 sec)
```

在设置 `innodb_file_per_table` 参数之前，需要先关闭数据库，然后在 `my.cnf` 文件中设置或者添加该参数 `innodb_file_per_table=1`，并重启数据库才能生效。这时候查看多表存储空间模式是否已经开启：

```
mysql> show variables like '%per_table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| innodb_file_per_table | ON |
+-----+-----+
1 row in set (0.00 sec)
```

使用多表空间特性的表，可以比较方便地进行表备份和恢复操作，但是直接复制 `.idb` 文件是不行的，可以通过以下命令：

```
ALTER TABLE TABLE_NAME DISCARD TABLESPACE;
ALTER TABLE TABLE_NAME IMPORT TABLESPACE;
```

InnoDB 在功能上跟 MyISAM 存储引擎有很大的不同，在参数配置上，InnoDB 也是单独处理的，InnoDB 所有的参数基本上都是加了前缀 “`innodb_`”。

如果想屏蔽 InnoDB 存储引擎，在 `my.ini` 配置文件中，将 `skip-innodb` 参数前的 `#` 去除，这样就无法创建 InnoDB 类型的表了。

1.3.3 MEMORY

MEMORY 存储引擎通过采用内存中的内容来创建表。每个 MEMORY 表实际上和一个磁盘文件关联起来。文件名采用 “表名.frm” 的格式。MEMORY 类型的表访问速度非常快，因为数据来源于内存空间。MEMORY 存储引擎默认使用 HASH 索引，虽然 MEMORY 类型的表访问速度非常快，但是一旦数据库发生故障关闭，内存中的数据就会发生丢失。

下面创建一个 MEMORY 类型的表 `grade`，其执行效率非常快，命令如下：

```
mysql> create table grade (
-> id int
```

```

-> )engine = memory;
Query OK, 0 rows affected (0.13 sec)

mysql> insert into grade select *from grade;
Query OK, 1048832 rows affected (1.11 sec)
Records: 1048832 Duplicates: 0 Warnings: 0

mysql> select count(*) from grade;
+-----+
| count(*) |
+-----+
| 2097664 |
+-----+
1 row in set (0.06 sec)

mysql> show table status like 'grade' \g;
Name:                grade
Engine:              MEMORY
Version:             10
Row_format:          Fixed
Rows:                2097664
Avg_row_length:      5
Data_length:         16939528
Max_data_length:10485760
Index_length:        0
Data_free:           0
Auto_increment:      NULL
Create_time:         NULL
Update_time:         NULL
Check_time:          NULL
Collation:           latin1_swedish_ci
Checksum:            NULL
Create_options:
Commen:

```

MEMORY 表的空间以小块来分配。创建 MEMORY 表的时候，可以通过添加一个索引指定是 HASH 索引或者 BTREE 索引。

例如，以下实例中指定了索引类型为 HASH，命令如下：

```

mysql> create table table_memory(
-> id int,
-> index using hash(id)
-> )engine=memory;
Query OK, 0 rows affected (0.08 sec)

```

查看索引类型的命令如下：


```
mysql> show index from table_memory \g;
Table:                table_memory
Non_unique:            1
Key_name:              id
Seq_in_index:          1
Column_name:           id
Collation:             NULL
Cardinality:           0
Sub_part:              NULL
Packed:               NULL
NULL:                 YES
Index_type:            HASH
Comment:
```

可以通过如下命令删除 `table_memory` 的 HASH 索引：

```
mysql> drop index id on table_memory;
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

可以通过如下命令修改 `table_memory` 的索引：

```
mysql> create index id_index using btree on table_memory(id);
Query OK, 0 rows affected (0.00 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

索引修改后，可以查看修改的效果，命令如下：

```
mysql> show index from table_memory \g;
Table:                table_memory
Non_unique:            1
Key_name:              id_index
Seq_in_index:          1
Column_name:           id
Collation:             A
Cardinality:           NULL
Sub_part:              NULL
Packed:               NULL
NULL:                 YES
Index_type:            BTREE
Comment:
```

MEMORY 表内容存储在内存中，如果一个内部表变的很大，服务器自动把它转换成为一个磁盘表。尺寸限制由 `temp_table_size` 系统变量的值来确定。每个 MEMORY 表容量的大小可以通过设置 `max_heap_table_size` 变量的值来控制。`max_heap_table_size` 系统变量默认的值是 16MB，对于单个表，也可以在 CREATE TABLE 语句中指定一个 MAX_ROWS 表选项，参数 MAX_ROWS 指定表的最大行数。例如以下实例：

```
mysql> create table table_memory_1(
-> id int primary key
-> )engine=memory max_rows=10000;
Query OK, 0 rows affected (0.00 sec)
```

在 MySQL 服务器启动时可以使用 `init-file` 选项。可以把 `INSERT INTO ... SELECT` 或 `LOAD DATA INFILE` 这样的语句放入这个文件中，从而从数据源中装载数据表。

MEMORY 类型的存储引擎主要用于内容变化不频繁的代码表，对 MEMORY 存储引擎的表更新操作数据不会写入到磁盘文件中，所以在选择 MEMORY 存储引擎的时候需要考虑这一特性。

1.3.4 MERGE

MERGE 存储引擎是一组 MyISAM 表的组合，将一组结构相同的 MyISAM 表组合成一个逻辑单元，通常也叫做 MRG_MYISAM 存储引擎。MERGE 表本身没有数据，对于 MERGE 类型表的插入操作，是通过 `INSERT_METHOD` 子句完成，可以使用 `FIRST` 或者 `LAST` 值，可以使其数据增加到第一个表，或者最后一个表上。其实上述操作实际上是对内部 MyISAM 表进行操作，所以在创建 MERGE 表时，MySQL 只会生成两个较小的文件，一个是 `.frm` 的文件，用于存放数据，还有一个 `.MRG` 文件，用于存放 MERGE 表的名称，包括 MERGE 表由哪些表组成。

下面是一个创建和使用 MERGE 表的例子：

步骤 01 创建三个表 `table_myisam_1`，`table_myisam_2`，`table_merge_12`（MERGE 表）。

```
mysql> create table table_myisam_1(
-> id int primary key,.
-> data datetime
-> )engine=myisam default charset=gbk;
Query OK, 0 rows affected (0.08 sec)

mysql> create table table_myisam_2(
-> id int primary key,
-> data datetime
-> )engine=myisam default charset=gbk;
Query OK, 0 rows affected (0.00 sec)

mysql> create table table_merge_12(
-> id int primary key,
-> data datetime
-> )engine=merge union=(table_myisam_1,table_myisam_2)
insert_method=first;
Query OK, 0 rows affected (0.03 sec)
```

步骤 02 向前两个表 `table_myisam_1`，`table_myisam_2` 添加数据。


```
mysql> insert into table_myisam_1 values(1,'2012-1-2'),(2,'2012-1-3');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

```
mysql> insert into table_myisam_2 values(1,'2012-1-2'),(2,'2012-1-3');
Query OK, 2 rows affected (0.02 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

步骤 03 查询 MERGE 表 table_merge_12 的数据。

```
mysql> select *from table_merge_12;
+----+-----+
| id | data                |
+----+-----+
| 1  | 2012-01-02 00:00:00 |
| 2  | 2012-01-03 00:00:00 |
| 1  | 2012-01-02 00:00:00 |
| 2  | 2012-01-03 00:00:00 |
+----+-----+
4 rows in set (0.00 sec)
```

步骤 04 向 MERGE 表 table_merge_12 添加一条数据。

```
mysql> insert into table_merge_12 values(3,'2020-2-3');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select *from table_myisam_1;
+----+-----+
| id | data                |
+----+-----+
| 1  | 2012-01-02 00:00:00 |
| 2  | 2012-01-03 00:00:00 |
| 3  | 2020-02-03 00:00:00 |
+----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select *from table_myisam_2;
+----+-----+
| id | data                |
+----+-----+
| 1  | 2012-01-02 00:00:00 |
| 2  | 2012-01-03 00:00:00 |
+----+-----+
2 rows in set (0.00 sec)
```

此时会发现 insert_method=first 起作用了，数据已经添加到前一个表中，merge 表还是前两个表合并的结果，通常如果没有指定 insert_method 参数，任何尝试往 MERGE 表中 INSERT

数据的操作，都会发生错误。通常使用 MERGE 表来透明地对多个表进行查询和更新操作，如果要了解 MERGE 表的更多信息，请参看 `sql/ha_myisammrg.h/ha_myisammrg.cc` 以及 `storage/myisammrg` 目录中的.c 文件。

1.3.5 BerkeleyDB 存储引擎

BerkeleyDB 存储引擎不是 MySQL 开发的存储引擎，是由 Sleepycat Software 开发的事务性存储引擎，简称为 BDB。

创建 BDB 表会生成两个数据文件，文件的名称用表名来开头，扩展名表示其不同的文件类型，一个.frm 文件存储表元数据，另一个.db 文件包含数据和索引内容。另外，BDB 为了实现事务安全也有自己的 redo 日志，可以通过参数设置日志存放的位置，在锁定机制方面，BDB 和 MEMORY 存储引擎一样，实现页级锁定。

创建一个 BDB 表，可以用 ENGINE 或者 TYPE 表选项来创建，命令如下：

```
mysql> create table table_bdb_1(
-> id int primary key,
-> data varchar(20)
-> )engine = BDB;
Query OK, 0 rows affected, 1 warning (0.05 sec)
```

查看创建表的状态，命令如下：

```
mysql> show table status like 'table_bdb_1';
Name:                table_bdb_1
Engine:              BerkeleyDB
Version:             10
Row_format:          Dynamic
Rows:                0
Avg_row_length:      0
Data_length:         0
Max_data_length:     281474976710655
Index_length:        1024
Data_free:           0
Auto_increment:      NULL
Create_time:         2012-06-02 23:01:02
Update_time:         2012-06-02 23:01:02
Check_time:          NULL
Collation:           latin1_swedish_ci
Checksum:            NULL
Create_options:
Comment:
```

由于 BDB 存储引擎实现了事务安全，BDB 在每次启动的时候，都会做一次检查操作，并且将所有的 redo 日志清空。

1.4 MySQL 工具

MySQL 数据库管理系统提供了许多命令行工具，这些工具可以用来管理 MySQL 服务器、对数据库进行访问控制、管理 MySQL 用户以及数据库备份和恢复工具等。而且 MySQL 提供图形化的管理工具，这使得对数据库的操作更加简单。本节将为读者介绍这些工具的作用。

1.4.1 MySQL 命令行实用程序

MySQL 服务器端实用工具程序如下：

(1) `mysqld`：SQL 后台程序（即 MySQL 服务器进程）。该程序必须运行之后，客户端才能通过连接服务器来访问数据库。

(2) `mysqld_safe`：服务器启动脚本。在 Unix 和 NetWare 中推荐使用 `mysqld_safe` 来启动 `mysqld` 服务器。`mysqld_safe` 增加了一些安全特性，例如当出现错误时重启服务器并向错误日志文件写入运行时间信息。

(3) `mysql.server`：服务器启动脚本。该脚本用于使用包含为特定级别的、运行启动服务脚本的、运行目录的系统。它调用 `mysqld_safe` 来启动 MySQL 服务器。

(4) `mysqld_multi`：服务器启动脚本，可以启动或停止系统上安装多个服务器。

(5) `myisamchk`：用来描述、检查、优化和维护 MyISAM 表的实用工具。

(6) `mysql.server`：服务器启动脚本。在 Unix 中的 MySQL 分发版包括 `mysql.server` 脚本。

(7) `mysqlbug`：MySQL 缺陷报告脚本。它可以用来向 MySQL 邮件系统发送缺陷报告。

(8) `mysql_install_db`：该脚本用默认权限创建 MySQL 授权表。通常只是在系统上首次安装 MySQL 时执行一次。

MySQL 客户端实用工具程序如下：

(1) `myisampack`：压缩 MyISAM 表以产生更小的只读表的一个工具。

(2) `mysql`：交互式输入 SQL 语句或从文件以批处理模式执行它们的命令行工具。

(3) `mysqlaccess`：检查访问主机名、用户名和数据库组合的权限的脚本。

(4) `mysqladmin`：执行管理操作的客户程序，例如创建或删除数据库，重载授权表，将表刷新到硬盘上，以及重新打开日志文件。`mysqladmin` 还可以用来检索版本、进程，以及服务器的状态信息。

(5) `mysqlbinlog`：从二进制日志读取语句的工具。在二进制日志文件中包含执行过的语句，可用来帮助系统从崩溃中恢复。

(6) `mysqlcheck`：检查、修复、分析以及优化表的表维护客户程序。

(7) `mysqldump`：将 MySQL 数据库转储到一个文件（例如 SQL 语句或 tab 分隔符文本文件）的客户程序。

(8) `mysqlhotcopy`：当服务器在运行时，快速备份 MyISAM 或 ISAM 表的工具。

(9) `mysql import`: 使用 `LOAD DATA INFILE` 将文本文件导入相关表的客户程序。

(10) `mysqlshow`: 显示数据库、表、列以及索引相关信息的客户程序。

(11) `pererror`: 显示系统或 MySQL 错误代码含义的工具。

下面挑选一个常用的命令工具以案例的形式讲解, 其他的工具的使用方法本书的其他章节会介绍。

1. MySQL 客户端连接工具——mysql

在数据库管理工具中, 经常使用 MySQL 客户端连接工具, 下面介绍 MySQL 命令的语法:

```
mysql [options] [database]
```

连接选项的含义如下:

- `-user`: 简称为 `-u`, 指定数据库用户名。
- `-password`: 简称为 `-p`, 指定登录密码。
- `-host`: 简称为 `-h`, 指定远程 MySQL 服务器的 ip 地址。
- `-port`: 简称为 `-P`, 指定连接的端口。

下面以登录本地 MySQL 服务器为例进行讲解, 操作命令如下:

```
[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.0.89-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

同样, 登录远程 MySQL 服务器的命令如下:

```
C:\>mysql -u ivan -h 192.168.0.5 -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.0.89-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

登录成功后, 可以通过如下命令查看当前连接用户:

```
mysql> select current_user();
+-----+
| current_user() |
```

```

+-----+
| ivan@% |
+-----+
1 row in set (0.08 sec)

```

提示

在正式生产环境之后，DBA 往往创建应用账户并赋予一定的权限，而不用 root 用户直接登录。MySQL 数据库默认的端口是 3306，建议在安装的时候修改 MySQL 端口，可以修改为其他任意系统没使用过的端口。

MySQL 的字符集选项可以在/etc/my.cnf 中配置，也可以在 MySQL 命令中手工指定客户端字符集，操作命令如下所示：

```

[root@localhost ~]# mysql -u root -p --default-character-set=gbk
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.0.89-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show variables like 'char%';
character_set_client:gbk
character_set_connection:gbk
character_set_database:latin1
character_set_filesystem:binary
character_set_results:gbk
character_set_server:latin1
character_set_system:utf8
character_sets_dir:/usr/local/mysql /share/mysql/charsets/
8 rows in set (0.15 sec)

```

mysql 工具提供了可以直接在 MySQL 客户端执行的语句，对于一些批处理脚本，这种方式比较方便，下面是一个执行 sql 语句并退出的例子。

```

[root@localhost ~]# mysql -u root -p mysql -e "select host,user from user";
Enter password:
+-----+-----+
| host          | user          |
+-----+-----+
| localhost     | root          |
| localhost.localdomain | root          |
| 127.0.0.1     | root          |
| localhost     |               |
| localhost.localdomain |               |
| %             | ivan          |

```

提示

-e 选项后面是可以执行的 sql 语句，-p 选项表示要求输入数据库密码，-p 后面的 mysql 表示具体的 mysql 数据库，查询结果中 host 表示用户远程允许访问的 ip 地址，%表示任何 ip 地址都可以访问。

2. MyISAM 表压缩工具——myisampack

使用 myisampack 工具可以压缩 MyISAM 表，压缩之后的表将占更少的磁盘存储空间，但是压缩之后的表是一个只读表。通常情况下，myisampack 可以将数据文件压缩到 40%~70% 左右，当以后使用表时，解压缩之后的信息会读入到内存之中。

使用 myisampack 的语法如下：

```
myisampack [options] filename ...
```

下面是一个 myisampack 压缩 MyISAM 表的例子：

```
[root@localhost books]# cd /usr/local/mysql/data/books;
[root@localhost books]# ls -al | grep books;
-rw-rw---- 1 mysql mysql      8586 2014-06-04 15:00 books.frm
-rw-rw---- 1 mysql mysql 10485760 2014-06-04 15:02 books.MYD
-rw-rw---- 1 mysql mysql  5390336 2014-06-04 15:02 books.MYI
```

可以看出 books 表的数据文件 books.MYD 大小是 10485760 字节，books.MYI 索引文件大小是 5390336 字节，下面就进行压缩操作：

```
[root@localhost books]# myisampack books;
Compressing books.MYD: (524288 records)
- Calculating statistics
- Compressing file
70.3%
Remember to run myisamchk -rq on compressed tables
[root@localhost books]# ls -al |grep books;
-rw-rw---- 1 mysql mysql      8586 2012-06-04 15:00 books.frm
-rw-rw---- 1 mysql mysql 3114102 2012-06-04 15:02 books.MYD
-rw-rw---- 1 mysql mysql     1024 2012-06-04 15:03 books.MYI
[root@localhost books]#
```

可以看出 books 表压缩之后的大小已经小了很多，值得注意的是，通常压缩 MyISAM 表之后的表是一个只读表，不能进行 DML 操作。

下面可以更新一条语句试试，此时会弹出错误提示。

首先查看 books 表中 id 为 1 的记录，命令如下：

```
mysql> select * from books where id = 1;
+-----+
| id | name |
```



```
+-----+-----+
| 1 | book1 |
+-----+-----+
1 row in set (0.29 sec)
```

更新 id 为 1 的记录，弹出错误提示信息：

```
mysql> update books set name = 'books2' where id = 1;
ERROR 1036 (HY000): Table 'books' is read only
```

3. MySQL 管理工具——mysqladmin

mysqladmin 是一个执行管理操作的客户程序。可以用它来检查服务器的配置和当前的状态，创建并删除数据库等等。

mysqladmin 工具的一些参数和含义如下所示：

```
Usage: mysqladmin [OPTIONS] command command....
--default-character-set=name  设置默认的字符集
-h, --host=name              连接主机
-p, --password[=name]        当连接 MySQL 数据库时需要使用的密码
-P, --port=#                 连接 MySQL 时所用的端口，默认是3306
--protocol=name              连接数据库
-O, --set-variable=name      登录数据时可以修改的一些参数信息
-u, --user=name              登录时使用的账户信息
--connect_timeout=#
--shutdown_timeout=#
```

通常使用 mysqladmin -u root -p shutdown 关闭数据库，下面是一些 mysqladmin 使用的例子：

(1) mysqladmin 可以用来创建数据库，如下所示：

```
[root@localhost ~]# mysqladmin -u root -p create t_db1
Enter password:
```

(2) 数据库创建完成之后，登录 mysql 可以查询出来：

```
mysql> show databases;
+-----+-----+
| Database          |
+-----+-----+
| information_schema |
| books              |
| database           |
| databse            |
| mysql              |
| t_db1              |
| test               |
```

```
| tt_db |
+-----+
8 rows in set (0.05 sec)
```

(3) `mysqladmin` 也可以用来删除数据库，命令如下所示：

```
[root@localhost ~]# mysqladmin -u root -p drop tt_db
Enter password:
Dropping the database is potentially a very bad thing to do.
Any data stored in the database will be destroyed.

Do you really want to drop the 'tt_db' database [y/N] y
Database "tt_db" dropped
```

通过上面的操作，数据库 `tt_db` 已经被删除了。

4. MySQL 日志管理工具——`mysqlbinlog`

MySQL 服务器生成的日志是以二进制文件格式存储在磁盘空间上，`mysqlbinlog` 工具的主要作用就是用来查看服务器上的日志文件的，`mysqlbinlog` 的使用的格式如下：

```
Shell> mysqlbinlog [options] log-files
```

下面通过一个案例来讲解如何使用 `mysqlbinlog` 工具查询日志文件。

步骤 01 首先把日志文件删除掉，删除日志之前查看下日志信息：

```
mysql> system ls -ltr mysql-bin*
-rw-rw---- 1 mysql mysql 639904 2015-06-02 07:21 mysql-bin.000002
-rw-rw---- 1 mysql mysql 15221 2015-06-02 07:21 mysql-bin.000001
-rw-rw---- 1 mysql mysql 279 2015-06-02 08:07 mysql-bin.000003
-rw-rw---- 1 mysql mysql 117 2015-06-02 08:09 mysql-bin.000004
-rw-rw---- 1 mysql mysql 117 2015-06-02 08:49 mysql-bin.000005
-rw-rw---- 1 mysql mysql 117 2015-06-02 09:23 mysql-bin.000006
-rw-rw---- 1 mysql mysql 117 2015-06-02 09:33 mysql-bin.000007
-rw-rw---- 1 mysql mysql 446 2015-06-02 10:56 mysql-bin.000008
-rw-rw---- 1 mysql mysql 2706 2015-06-02 19:50 mysql-bin.000009
-rw-rw---- 1 mysql mysql 9550 2015-06-02 23:59 mysql-bin.000010
-rw-rw---- 1 mysql mysql 465 2015-06-03 00:14 mysql-bin.000011
-rw-rw---- 1 mysql mysql 2946 2015-06-04 15:02 mysql-bin.000012
-rw-rw---- 1 mysql mysql 247 2015-06-04 15:30 mysql-bin.index
-rw-rw---- 1 mysql mysql 898 2015-06-04 16:53 mysql-bin.000013
```

步骤 02 执行 `reset master` 命令删除日志文件：

```
mysql> reset master;
Query OK, 0 rows affected (0.05 sec)
```

步骤 03 接着往测试表 `emp` 中插入一条数据，再删除表中所有数据：

```
mysql> insert into emp values(4,'kk');
Query OK, 1 row affected (0.01 sec)

mysql> delete from emp;
Query OK, 4 rows affected (0.00 sec)
```

步骤 04 下面使用 mysqlbinlog 工具查询日志文件，如下所示：

```
[root@localhost data]# mysqlbinlog mysql-bin.000001
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#150604 17:12:21 server id 1 end_log_pos 98      Start: binlog v 4, server v
5.0.89-log created 150604 17:12:21 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
# at 98
#150604 17:18:08 server id 1 end_log_pos 192  Query  thread_id=18
exec_time=0      error_code=0
use t_db1/*!*/;
SET TIMESTAMP=1338844688/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1,
@@session.unique_checks=1/*!*/;
SET @@session.sql_mode=0/*!*/;
/*!\C latin1 *//*!*/;
SET
@@session.character_set_client=8,@@session.collation_connection=8,@@session.co
llation_server=8/*!*/;
insert into emp values(4,'kk')
/*!*/;
# at 192
#150604 17:18:15 server id 1 end_log_pos 271  Query  thread_id=18
exec_time=0 error_code=0
SET TIMESTAMP=1338844695/*!*/;
delete from emp
/*!*/;
DELIMITER ;
# End of log file
ROLLBACK /* added by mysqlbinlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
[root@localhost data]#
```

通过以上步骤可以查看到数据操作的历史信息。接下来讲解 mysqlbinlog 日志管理工具的参数选项的用法。

(1) -d 参数：日志文件只列出指定数据库的相关操作。下面通过案例来讲解，命令如下：

```
[root@localhost data]# mysqlbinlog mysql-bin.000001 -d books;
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#150604 17:12:21 server id 1 end_log_pos 98      Start: binlog v 4, server v
5.0.89-log created 120604 17:12:21 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
# at 98
# at 192
# at 271
# at 365
# at 460
#150604 17:41:40 server id 1 end_log_pos 540    Query    thread_id=18
exec_time=0      error_code=0
use books/*!*/;
SET TIMESTAMP=1338846100/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1,
@@session.unique_checks=1/*!*/;
SET @@session.sql_mode=0/*!*/;
/*!\C latin1 *//*!*/;
SET
@@session.character_set_client=8,@@session.collation_connection=8,@@session.co
llation_server=8/*!*/;
drop table books
/*!*/;
DELIMITER ;
# End of log file
ROLLBACK /* added by mysqlbinlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
[root@localhost data]#
```

(2) --start-datetime 参数：指定从某个日期开始的相关数据操作。

(3) --stop-datetime 参数：指定从某个日期结束的相关数据操作。

下面通过案例来讲解上述两个参数的使用方法。

```
[root@localhost data]# mysqlbinlog mysql-bin.000001
--start-datetime='2015/06/01 06:00:00' --stop-datetime='2021/06/04 08:00:00'
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
```

```

#150604 17:12:21 server id 1 end_log_pos 98      Start: binlog v 4, server v
5.0.89-log created 120604 17:12:21 at startup
# Warning: this binlog is either in use or was not closed properly.
ROLLBACK/*!*/;
# at 98
#150604 17:18:08 server id 1 end_log_pos 192      Query    thread_id=18
exec_time=0 error_code=0
use t_db1/*!*/;
SET TIMESTAMP=1338844688/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1,
@@session.unique_checks=1/*!*/;
SET @@session.sql_mode=0/*!*/;
/*!\C latin1 *//*!*/;
SET
@@session.character_set_client=8,@@session.collation_connection=8,@@session.co
llation_server=8/*!*/;
insert into emp values(4,'kk')
/*!*/;
# at 192
#150604 17:18:15 server id 1 end_log_pos 271      Query    thread_id=18
exec_time=0 error_code=0
SET TIMESTAMP=1338844695/*!*/;
delete from emp
/*!*/;
# at 271
#150604 17:35:46 server id 1 end_log_pos 365      Query    thread_id=18
exec_time=0 error_code=0
SET TIMESTAMP=1338845746/*!*/;
insert into emp values(4,'kk')
/*!*/;
# at 365
#150604 17:35:53 server id 1 end_log_pos 460      Query    thread_id=18
exec_time=0 error_code=0
SET TIMESTAMP=1338845753/*!*/;
insert into emp values(5,'kka')
/*!*/;
# at 460
#150604 17:41:40 server id 1 end_log_pos 540      Query    thread_id=18
exec_time=0 error_code=0
use books/*!*/;
SET TIMESTAMP=1338846100/*!*/;
drop table books
/*!*/;
DELIMITER ;
# End of log file

```



```
ROLLBACK /* added by mysqlbinlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
[root@localhost data]#
```

5. MyISAM 表维护工具——mysqlcheck

mysqlcheck 的功能类似 myisamchk，但其工作时间不同。主要差别是当 mysqld 服务器在运行时必须使用 mysqlcheck，而 myisamchk 应用于服务器没有运行时。使用 mysqlcheck 的好处是不需要停止服务器来检查或修复表，另外使用 myisamchk 修复失败是不可逆的。

有 3 种方式可以调用 mysqlcheck，命令分别如下：

```
mysqlcheck[options] db_name [tables]
mysqlcheck[options] --database DB1 [DB2 DB3...]
mysqlcheck[options] --all--database
```

options 参数最常用的有以下几个选项：

- -all-databases: 简称为-A，用于检测所有的数据库。
- -analyze: 简称为-a，用于分析指定的数据库。
- -auto-repair: 如果检测的数据表发生损坏，将会在所有表检测完后自动修复它。
- -check: 简称为-c，用于检测表的错误。
- -repair: 简称为-r，该参数用来修复任何错误，除非唯一键不是唯一的。
- -optimize: 简称为-o，用于优化指定的数据库。

下面将通过案例来理解上述参数的使用方法。

(1) 检测所有的数据库，操作命令如下：

```
[root@localhost data]# mysqlcheck -u root -p -A
Enter password:
books.authors OK
books.grade
note : The storage engine for the table doesn't support check
books.sclass OK
books.st OK
books.table_bdb_1 OK
books.table_bdb_2 OK
books.table_example OK
books.table_memory
note : The storage engine for the table doesn't support check
books.table_memory_1
note : The storage engine for the table doesn't support check
books.table_merge_12 OK
books.table_myisam_1 OK
books.table_myisam_2 OK
mysql.columns_priv OK
```

```

mysql.db OK
mysql.func OK
mysql.help_category OK
mysql.help_keyword OK
mysql.help_relation OK
mysql.help_topic OK
mysql.host OK
mysql.proc OK
mysql.procs_priv OK
mysql.tables_priv OK
mysql.time_zone OK
mysql.time_zone_leap_second OK
mysql.time_zone_name OK
mysql.time_zone_transition OK
mysql.time_zone_transition_type OK
mysql.user OK
t_db1.emp OK
[root@localhost data]#

```

(2) 检测指定的数据库，操作命令如下：

```

[root@localhost data]# mysqlcheck -u root -p -c mysql
Enter password:
mysql.columns_priv OK
mysql.db OK
mysql.func OK
mysql.help_category OK
mysql.help_keyword OK
mysql.help_relation OK
mysql.help_topic OK
mysql.host OK
mysql.proc OK
mysql.procs_priv OK
mysql.tables_priv OK
mysql.time_zone OK
mysql.time_zone_leap_second OK
mysql.time_zone_name OK
mysql.time_zone_transition OK
mysql.time_zone_transition_type OK
mysql.user OK

```

(3) 检测指定的数据库，如果有错误将自动修复，操作命令如下：

```

[root@localhost data]# mysqlcheck -u root -p -auto-repair mysql
Enter password:
mysql.columns_priv OK
mysql.db OK

```

```

mysql.func OK
mysql.help_category OK
mysql.help_keyword OK
mysql.help_relation OK
mysql.help_topic OK
mysql.host OK
mysql.proc OK
mysql.procs_priv OK
mysql.tables_priv OK
mysql.time_zone OK
mysql.time_zone_leap_second OK
mysql.time_zone_name OK
mysql.time_zone_transition OK
mysql.time_zone_transition_type OK
mysql.user OK

```

(4) 修复指定的数据库，操作命令如下：

```

[root@localhost data]# mysqlcheck -u root -p -r mysql
Enter password:
mysql.columns_priv OK
mysql.db OK
mysql.func OK
mysql.help_category OK
mysql.help_keyword OK
mysql.help_relation OK
mysql.help_topic OK
mysql.host OK
mysql.proc OK
mysql.procs_priv OK
mysql.tables_priv OK
mysql.time_zone OK
mysql.time_zone_leap_second OK
mysql.time_zone_name OK
mysql.time_zone_transition OK
mysql.time_zone_transition_type OK
mysql.user OK
[root@localhost data]#

```

(5) 分析指定的数据库，命令如下：

```

[root@localhost data]# mysqlcheck -u root -p -a mysql
Enter password:
mysql.columns_priv Table is already up to date
mysql.db Table is already up to date
mysql.func Table is already up to date
mysql.help_category Table is already up to date

```


| | |
|---------------------------------|-----------------------------|
| mysql.help_keyword | Table is already up to date |
| mysql.help_relation | Table is already up to date |
| mysql.help_topic | Table is already up to date |
| mysql.host | Table is already up to date |
| mysql.proc | Table is already up to date |
| mysql.procs_priv | Table is already up to date |
| mysql.tables_priv | Table is already up to date |
| mysql.time_zone | Table is already up to date |
| mysql.time_zone_leap_second | Table is already up to date |
| mysql.time_zone_name | Table is already up to date |
| mysql.time_zone_transition | Table is already up to date |
| mysql.time_zone_transition_type | Table is already up to date |
| mysql.user | Table is already up to date |

(6) 优化指定的数据库，操作命令如下：

```
[root@localhost data]# mysqlcheck -u root -p -o mysql
Enter password:
mysql.columns_priv      OK
mysql.db                OK
mysql.func              OK
mysql.help_category     OK
mysql.help_keyword      OK
mysql.help_relation     OK
mysql.help_topic        OK
mysql.host              OK
mysql.proc              OK
mysql.procs_priv        OK
mysql.tables_priv       OK
mysql.time_zone         OK
mysql.time_zone_leap_second OK
mysql.time_zone_name    OK
mysql.time_zone_transition OK
mysql.time_zone_transition_type OK
mysql.user              OK
```

1.4.2 MySQL Workbench

MySQL Workbench 是下一代可视化数据库设计软件，MySQL Workbench 为数据库管理员和开发人员提供了一整套可视化数据库操作环境，主要功能有：

- 数据库设计和模型建立。
- SQL 开发（取代 MySQL Query Browser）。
- 数据库管理（取代 MySQL Administrator）。

MySQL Workbench 有两个版本：

(1) MySQL Workbench Community Edition (也叫 MySQL Workbench OSS, 社区版), MySQL Workbench OSS 是在 GPL 证书下发布的开源社区版本。

(2) MySQL Workbench Standard Edition (也叫 MySQL Workbench SE, 商业版), MySQL Workbench SE 是按年收费的商业版本。

提示

截至本书完稿时, 最新版本为 MySQL Workbench 6.3.3。一些出版时间较早的 MySQL 教程中会提到图形化的工具 MySQL Query Browser 和 MySQL Administrator。随着 MySQL 的发展, MySQL 使用更高效、便捷的 Workbench 替换掉了这两个工具, 目前官方已经不再提供 MySQL Query Browser 和 MySQL Administrator 的技术支持和更新, 但是仍然可以在使用旧版本 MySQL 时使用它们。

1.5 本章小结

本章主要讲解了 MySQL 数据库的逻辑结构, 重点讲解了 MySQL 提供的几种主要的存储引擎及其特性, 以及介绍了每种存储引擎的优势和其主要适用的地方, 本章也介绍了 MySQL 数据库提供的常用的工具, 其中数据库备份工具, 导入导出工具将在后面章节涉及。通过本章的学习, 读者可以初步了解 MySQL 数据的整体的逻辑框架, 对 MySQL 数据库有一个整体的认识。

第 2 章

◀ MySQL 权限与安全 ▶

对于企业而言，数据库中保存的企业业务数据是非常重要的信息，尤其是互联网企业，数据库中的用户信息是企业的根本资源。MySQL 数据库管理系统的安全性涉及方方面面，不仅和操作系统本身有很大的关系，还和数据库本身的权限系统有很大的关系。MySQL 的权限系统主要用来检测数据库用户是否属于合法的用户，如果是合法的用户，则赋予相应的数据库使用权限。数据库的安全性在很大程度上跟数据库的权限系统有很大的关系，不当的权限设置可能会导致各种各样的安全隐患。本章主要讲述数据库的安全问题和 MySQL 权限系统的设置等。

2.1 权限表

MySQL 服务器通过权限表来控制用户对数据库的访问，权限表存放在 mysql 数据库中，由 mysql_install_db 脚本初始化。存储账户权限信息表主要有：user、db、host、tables_priv、columns_priv 和 procs_priv。本节将为读者介绍这些表的内容和作用。

2.1.1 user 表

user 表是 MySQL 中最重要的一個权限表，记录允许连接到服务器的账号信息，里面的权限是全局级的。例如：一个用户在 user 表中被授予了 DELETE 权限，则该用户可以删除 MySQL 服务器上所有数据库中的任何记录。MySQL 5.6 中 user 表有 42 个字段，如表 2-1 所示，这些字段可以分为 4 类，分别是用户列、权限列、安全列和资源控制列。本节将为读者介绍 user 表中各字段的含义。

表 2-1 user 表结构

| 字段名 | 数据类型 | 默认值 |
|-------------|---------------|-----|
| Host | char(60) | |
| User | char(16) | |
| Password | char(41) | |
| Select_priv | enum('N','Y') | N |
| Insert_priv | enum('N','Y') | N |
| Update_priv | enum('N','Y') | N |

(续表)

| 字段名 | 数据类型 | 默认值 |
|------------------------|-----------------------------------|------|
| Delete_priv | enum('N','Y') | N |
| Create_priv | enum('N','Y') | N |
| Drop_priv | enum('N','Y') | N |
| Reload_priv | enum('N','Y') | N |
| Shutdown_priv | enum('N','Y') | N |
| Process_priv | enum('N','Y') | N |
| File_priv | enum('N','Y') | N |
| Grant_priv | enum('N','Y') | N |
| References_priv | enum('N','Y') | N |
| Index_priv | enum('N','Y') | N |
| Alter_priv | enum('N','Y') | N |
| Show_db_priv | enum('N','Y') | N |
| Super_priv | enum('N','Y') | N |
| Create_tmp_table_priv | enum('N','Y') | N |
| Lock_tables_priv | enum('N','Y') | N |
| Execute_priv | enum('N','Y') | N |
| Repl_slave_priv | enum('N','Y') | N |
| Repl_client_priv | enum('N','Y') | N |
| Create_view_priv | enum('N','Y') | N |
| Show_view_priv | enum('N','Y') | N |
| Create_routine_priv | enum('N','Y') | N |
| Alter_routine_priv | enum('N','Y') | N |
| Create_user_priv | enum('N','Y') | N |
| Event_priv | enum('N','Y') | N |
| Trigger_priv | enum('N','Y') | N |
| Create_tablespace_priv | enum('N','Y') | N |
| ssl_type | enum('','ANY','X509','SPECIFIED') | |
| ssl_cipher | blob | NULL |
| x509_issuer | blob | NULL |
| x509_subject | blob | NULL |
| max_questions | int(11) unsigned | 0 |
| max_updates | int(11) unsigned | 0 |
| max_connections | int(11) unsigned | 0 |
| max_user_connections | int(11) unsigned | 0 |
| plugin | char(64) | |
| authentication_string | text | NULL |

1. 用户列

user 表的用户列包括 Host、User、Password，分别表示主机名、用户名和密码。其中 User 和 Host 为 User 表的联合主键。当用户与服务器之间建立连接时，输入的账户信息中的用户名

称、主机名和密码必须匹配 `User` 表中对应的字段，只有 3 个值都匹配的时候，才允许连接的建立。这 3 个字段的值就是创建账户时保存的账户信息。修改用户密码时，实际就是修改 `user` 表的 `Password` 字段的值。

2. 权限列

权限列的字段决定了用户的权限，描述了在全局范围内允许对数据和数据库进行的操作。包括查询权限、修改权限等普通权限，还包括了关闭服务器、超级权限和加载用户等高级权限。普通权限用于操作数据库；高级权限用于数据库管理。

`user` 表中对应的权限是针对所有用户数据库的。这些字段值的类型为 `ENUM`，可以取的值只能为 `Y` 和 `N`，`Y` 表示该用户有对应的权限；`N` 表示用户没有对应的权限。查看 `user` 表的结构可以看到，这些字段的值默认都是 `N`。如果要修改权限，可以使用 `GRANT` 语句或 `UPDATE` 语句更改 `user` 表的这些字段来修改用户对应的权限。

3. 安全列

安全列只有 6 个字段，其中两个是 `ssl` 相关的，2 个是 `x509` 相关的，另外 2 个是授权插件相关的。`ssl` 用于加密；`X509` 标准可用于标识用户；`Plugin` 字段标识可以用于验证用户身份的插件，如果该字段为空，服务器使用内建授权验证机制验证用户身份。读者可以通过 `SHOW VARIABLES LIKE 'have_openssl'` 语句来查询服务器是否支持 `ssl` 功能。

4. 资源控制列

资源控制列的字段用来限制用户使用的资源，包含 4 个字段，分别为：

- `max_questions`——用户每小时允许执行的查询操作次数。
- `max_updates`——用户每小时允许执行的更新操作次数。
- `max_connections`——用户每小时允许执行的连接操作次数。
- `max_user_connections`——用户允许同时建立的连接次数。

一个小时内用户查询或者连接数量超过资源控制限制，用户将被锁定，直到下一个小时，才可以在此执行对应的操作。可以使用 `GRANT` 语句更新这些字段的值。

2.1.2 db 表和 host 表

`db` 表和 `host` 表是 MySQL 数据中非常重要的权限表。`db` 表中存储了用户对某个数据库的操作权限，决定用户能从哪个主机存取哪个数据库。`host` 表中存储了某个主机对数据库的操作权限，配合 `db` 权限表对给定主机上数据库级操作权限做更细致的控制。这个权限表不受 `GRANT` 和 `REVOKE` 语句的影响。`db` 表比较常用，`host` 表一般很少使用。`db` 表和 `host` 表结构相似，字段大致可以分为两类：用户列和权限列。`db` 表和 `host` 表的结构分别如表 2-2 和表 2-3 所示。

表 2-2 db 表结构

| 字段名 | 数据类型 | 默认值 |
|-----------------------|---------------|-----|
| Host | char(60) | |
| Db | char(64) | |
| User | char(16) | |
| Select_priv | enum('N','Y') | N |
| Insert_priv | enum('N','Y') | N |
| Update_priv | enum('N','Y') | N |
| Delete_priv | enum('N','Y') | N |
| Create_priv | enum('N','Y') | N |
| Drop_priv | enum('N','Y') | N |
| Grant_priv | enum('N','Y') | N |
| References_priv | enum('N','Y') | N |
| Index_priv | enum('N','Y') | N |
| Alter_priv | enum('N','Y') | N |
| Create_tmp_table_priv | enum('N','Y') | N |
| Lock_tables_priv | enum('N','Y') | N |
| Create_view_priv | enum('N','Y') | N |
| Show_view_priv | enum('N','Y') | N |
| Create_routine_priv | enum('N','Y') | N |
| Alter_routine_priv | enum('N','Y') | N |
| Execute_priv | enum('N','Y') | N |
| Event_priv | enum('N','Y') | N |
| Trigger_priv | enum('N','Y') | N |

表 2-3 host 表结构

| 字段名 | 数据类型 | 默认值 |
|-----------------------|---------------|-----|
| Host | char(60) | |
| Db | char(64) | |
| Select_priv | enum('N','Y') | N |
| Insert_priv | enum('N','Y') | N |
| Update_priv | enum('N','Y') | N |
| Delete_priv | enum('N','Y') | N |
| Create_priv | enum('N','Y') | N |
| Drop_priv | enum('N','Y') | N |
| Grant_priv | enum('N','Y') | N |
| References_priv | enum('N','Y') | N |
| Index_priv | enum('N','Y') | N |
| Alter_priv | enum('N','Y') | N |
| Create_tmp_table_priv | enum('N','Y') | N |
| Lock_tables_priv | enum('N','Y') | N |

(续表)

| 字段名 | 数据类型 | 默认值 |
|---------------------|---------------|-----|
| Create view priv | enum('N','Y') | N |
| Show view priv | enum('N','Y') | N |
| Create routine priv | enum('N','Y') | N |
| Alter routine priv | enum('N','Y') | N |
| Execute priv | enum('N','Y') | N |
| Trigger priv | enum('N','Y') | N |

1. 用户列

db 表用户列有 3 个字段，分别是 Host、User、Db，标识从某个主机连接某个用户对某个数据库的操作权限，这 3 个字段的组合构成了 db 表的主键。host 表不存储用户名称，用户列只有 2 个字段，分别是 Host 和 Db，表示从某个主机连接的用户对某个数据库的操作权限，其主键包括 Host 和 Db 两个字段。host 很少用到，一般情况下 db 表就可以满足权限控制需求了。

2. 权限列

db 表和 host 表的权限列大致相同，表中 create_routine_priv 和 alter_routine_priv 这两个字段表明用户是否有创建和修改存储过程的权限。

user 表中的权限是针对所有数据库的，如果希望用户只对某个数据库有操作权限，那么需要将 user 表中对应的权限设置为 N，然后在 db 表中设置对应数据库的操作权限。例如，有一个名称为 Zhangting 的用户分别从名称为 large.domain.com 和 small.domain.com 的两个主机连接到数据库，并需要操作 books 数据库。这时，可以将用户名称 Zhangting 添加到 db 表中，而 db 表中的 host 字段值为空，然后将两个主机地址分别作为两条记录的 host 字段值添加到 host 表中，并将两个表的数据库字段设置为相同的值 books。当有用户连接到 MySQL 服务器时，db 表中没有用户登录的主机名称，则 MySQL 会从 host 表中查找相匹配的值，并根据查询的结果决定用户的操作是否被允许。

2.1.3 tables_priv 表和 columns_priv 表

tables_priv 表用来对表设置操作权限，columns_priv 表用来对表的某一列设置权限。tables_priv 表和 columns_priv 表的结构分别如表 2-4 和表 2-5 所示。

表 2-4 tables_priv 表结构

| 字段名 | 数据类型 | 默认值 |
|------------|----------|-----|
| Host | char(60) | |
| Db | char(64) | |
| User | char(16) | |
| Table name | char(64) | |
| Grantor | char(77) | |

(续表)

| 字段名 | 数据类型 | 默认值 |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------|-------------------|
| Timestamp | timestamp | CURRENT_TIMESTAMP |
| Table_priv | set('Select','Insert','Update','Delete','Create','Drop','Grant','References','Index','Alter','Create View','Show view','Trigger') | |
| Column_priv | set('Select','Insert','Update','References') | |

表 2-5 columns_priv 表结构

| 字段名 | 数据类型 | 默认值 |
|-------------|----------------------------------------------|-------------------|
| Host | char(60) | |
| Db | char(64) | |
| User | char(16) | |
| Table_name | char(64) | |
| Column_name | char(64) | |
| Timestamp | timestamp | CURRENT_TIMESTAMP |
| Column_priv | set('Select','Insert','Update','References') | |

tables_priv 表有 8 个字段，分别是 Host、Db、User、Table_name、Grantor、Timestamp、Table_priv 和 Column_priv，各个字段说明如下：

- Host、Db、User 和 Table_name 4 个字段分别表示主机名、数据库名、用户名和表名。
- Grantor 表示修改该记录的用户。
- Timestamp 字段表示修改该记录的时间。
- Table_priv 表示对表的操作权限包括 Select、Insert、Update、Delete、Create、Drop、Grant、References、Index 和 Alter 等。
- Column_priv 字段表示对表中的列的操作权限，包括 Select Insert Update 和 References

Columns_priv 表只有 7 个字段，分别是 Host、Db、User、Table_name、Column_name、Timestamp、Column_priv。其中，Column_name 用来指定对哪些数据列具有操作权限。

2.1.4 procs_priv 表

procs_priv 表可以对存储过程和存储函数设置操作权限。Procs_priv 的表结构如表 2-6 所示。

表 2-6 procs_priv 表结构

| 字段名 | 数据类型 | 默认值 |
|--------------|------------------------------|------|
| Host | char(60) | |
| Db | char(64) | |
| User | char(16) | |
| Routine_name | char(64) | |
| Routine_type | enum('FUNCTION','PROCEDURE') | NULL |

(续表)

| 字段名 | 数据类型 | 默认值 |
|-----------|----------------------------------------|-------------------|
| Grantor | char(77) | |
| Proc_priv | set('Execute','Alter Routine','Grant') | |
| Timestamp | timestamp | CURRENT_TIMESTAMP |

procs_priv 表包含 8 个字段, 分别是 Host、Db、User、Routine_name、Routine_type、Grantor、Proc_priv 和 Timestamp, 各个字段的说明如下:

- Host、Db 和 User 字段分别表示主机名、数据库名和用户名。Routine_name 表示存储过程或函数的名称。
- Routine_type 表示存储过程或函数的类型。Routine_type 字段有两个值, 分别是 FUNCTION 和 PROCEDURE。FUNCTION 表示这是一个函数; PROCEDURE 表示这是一个存储过程。
- Grantor 是插入或修改该记录的用户。
- Proc_priv 表示拥有的权限, 包括 Execute、Alter Routine、Grant 3 种。
- Timestamp 表示记录更新时间。

提示

由于权限信息数据量比较小, 访问比较频繁, MySQL 在启动后会将权限表的信息缓存起来, 所以手工修改权限信息之后, 都需要执行“FLUSH PRIVILEGES”指令来重新刷新缓存中存储的权限信息。值得注意的是, 用户使用 grant/revoke/drop user/create user 命令修改用户权限的时候, 也会更新缓存中的权限信息。

2.2 账户管理

MySQL 提供许多语句用来管理用户账号, 这些语句可以用来管理包括登录和退出 MySQL 服务器、创建用户、删除用户、密码管理和权限管理等内容。MySQL 数据库的安全性, 需要通过账户管理来保证。本节将介绍 MySQL 中如何对账户进行管理。

2.2.1 登录和退出 MySQL 服务器

读者已经知道登录 MySQL 时, 使用 MySQL 命令并在后面指定登录主机以及用户名和密码。本小节将详细介绍 MySQL 命令的常用参数以及登录、退出 MySQL 服务器的方法。

通过 mysql -help 命令可以查看 mysql 命令帮助信息。MySQL 命令的常用参数如下:

- -h 主机名, 可以使用该参数指定主机名或 ip, 如果不指定, 默认是 localhost。
- -u 用户名, 可以使用该参数指定用户名。
- -p 密码, 可以使用该参数指定登录密码。如果该参数后面有一段字段, 则该段字符串

将作为用户的密码直接登录。如果后面没有内容，则登录的时候会提示输入密码。注意：该参数后面的字符串和-p 之前不能有空格。

- -P 端口号，该参数后面接 MySQL 服务器的端口号，默认为 3306。
- 数据库名，可以在命令的最后指定数据库名。
- -e 执行 SQL 语句。如果指定了该参数，将在登录后执行-e 后面的命令或 SQL 语句并退出。

【例 2.1】使用 root 用户登录到本地 MySQL 服务器的 test 库中，命令如下：

```
MySQL -h localhost -u root -p test
```

命令执行如下：

```
C:\> MySQL -h localhost -u root -p test
Enter password: **
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.6.10 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MySQL>
```

执行命令时，会提示 Enter password:，如果没有设置密码，可以直接按 Enter 键。输入密码正确就可以直接登录到服务器下面的 tes 数据库中了。

【例 2.2】使用 root 用户登录到本地 MySQL 服务器的 mysql 数据库中，同时执行一条查询语句。命令如下：

```
MySQL -h localhost -u root -p mysql -e "DESC person;"
```

命令执行如下：

```
C:\> MySQL -h localhost -u root -p mysql -e "DESC person;"
Enter password: **
+-----+-----+-----+-----+-----+-----+
+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
+
| id    | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name  | char(40)         | NO   |     |         |                |
```

```

+----+-----+-----+-----+-----+-----+
| age | int(11) | NO | 0 |
| info | char(50) | YES | NULL |
+----+-----+-----+-----+-----+-----+

```

按照提示输入密码，命令执行完成后查询出 `person` 表的结构，查询返回之后会自动退出 MySQL。

2.2.2 新建普通用户

创建新用户，必须有相应的权限来执行创建操作。在 MySQL 数据库中，有两种方式创建新用户：一种是使用 `CREATE USER` 或 `GRANT` 语句；另一种是直接操作 MySQL 授权表。最好的方法是使用 `GRANT` 语句，因为这样更精确，错误少。下面分别介绍这两种创建到用户的方法。

1. 使用 CREATE USER 语句创建新用户

执行 `CREATE USER` 或 `GRANT` 语句时，服务器会修改相应的用户授权表，添加或者修改用户及其权限。`CREATE USER` 语句的基本语法格式如下：

```

CREATE USER user_specification
[, user_specification] ...

user_specification:
    user@host
    [
        IDENTIFIED BY [PASSWORD] 'password'
        | IDENTIFIED WITH auth_plugin [AS 'auth_string']
    ]

```

`user` 表示创建的用户的名称；`host` 表示允许登录的用户主机名称；`IDENTIFIED BY` 表示用来设置用户的密码；`[PASSWORD]` 表示使用哈希值设置密码，该参数可选；`'password'` 表示用户登录时使用的普通明文密码；`IDENTIFIED WITH` 语句为用户指定一个身份验证插件；`auth_plugin` 是插件的名称，插件的名称可以是一个带单引号的字符串，或者带引号的字符串；`auth_string` 是可选的字符串参数，该参数将传递给身份验证插件，由该插件解释该参数的意义。

`CREATE USER` 语句会添加一个新的 MySQL 账户。使用 `CREATE USER` 语句的用户，必须有全局的 `CREATE USER` 权限或 MySQL 数据库的 `INSERT` 权限。每添加一个用户，`CREATE USER` 语句会在 `mysql.user` 表中添加一条新记录，但是新创建的账户没有任何权限。如果添加的账户已经存在，`CREATE USER` 语句会返回一个错误。

【例 2.3】 使用 `CREATE USER` 创建一个用户，用户名是 `jeffrey`，密码是 `mypass`，主机名是 `localhost`，命令如下：

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

如果只指定用户名部分'jeffrey'，主机名部分则默认为 '%'（即对所有的主机开放权限）。

user_specification 告诉 MySQL 服务器当用户登录时怎么验证用户的登录授权。如果指定用户登录不需要密码，可以省略 IDENTIFIED BY 部分：

```
CREATE USER 'jeffrey'@'localhost';
```

此种情况，MySQL 服务端使用内建的身份验证机制，用户登录时不能指定密码。

如果要创建指定密码的用户，需要 IDENTIFIED BY 指定明文密码值：

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'mypass';
```

此种情况，MySQL 服务端使用内建的身份验证机制，用户登录时必须指定密码。

为了避免指定明文密码，如果知道密码的散列值，可以通过 PASSWORD 关键字使用密码的哈希值设置密码。

密码的哈希值可以使用 password() 函数获取，如：

```
MySQL> SELECT password('mypass');
+-----+
| password('mypass') |
+-----+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-----+
1 row in set (0.00 sec)
```

*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 就是 mypass 的哈希值。接下来执行下面语句：

```
CREATE USER 'jeffrey'@'localhost'
IDENTIFIED BY PASSWORD '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

用户 jeffrey 的密码将被设定为 mypass。

对于使用插件认证连接的用户，服务器调用指定名称的插件，客户端需要提供验证方法所需要的凭据。如果创建用户时或者连接服务器时，服务器找不到对应的插件，将返回一个错误，IDENTIFIED WITH 语法格式如下：

```
CREATE USER 'jeffrey'@'localhost'
IDENTIFIED WITH my_auth_plugin;
```

提示

IDENTIFIED WITH 只能在 MySQL 5.5.7 及以上版本中使用。IDENTIFIED BY 和 IDENTIFIED WITH 是互斥的，所以对于一个账户来说只能使用一个验证方法。CREATE USER 语句的操作会被记录到服务器日志文件或者操作历史文件中，如 ~/.MySQL_history。这意味着对这些文件有读取权限的人，都可以读取新添加用户的明文密码。

MySQL 的某些版本中会引入授权表的结构变化，添加新的特权或功能。每当更新 MySQL 到一个新的版本时，应该更新授权表，以确保它们有最新的结构，确认可以使用任何新功能。

2. 使用 GRANT 语句创建新用户

CREATE USER 语句可以用来创建账户, 通过该语句可以在 user 表中添加一条新的记录, 但是 CREATE USER 语句创建的新用户没有任何权限, 还需要使用 GRANT 语句赋予用户权限。而 GRANT 语句不仅可以创建新用户, 还可以在创建的同时对用户授权。GRANT 还可以指定账户的其他特点, 如使用安全连接、限制使用服务器资源等。使用 GRANT 语句创建新用户时必须要有 GRANT 权限。GRANT 语句是添加新用户并授权他们访问 MySQL 对象的首选方法, GRANT 语句的基本语法格式如下:

```
GRANT privileges ON db.table
TO user@host [IDENTIFIED BY 'password'] [, user [IDENTIFIED BY 'password']]
[WITH GRANT OPTION];
```

其中, privileges 表示赋予用户的权限类型; db.table 表示用户的权限所作用的数据库中的表; IDENTIFIED BY 关键字用来设置密码; 'password' 表示用户密码; WITH GRANT OPTION 为可选参数, 表示对新建立的用户赋予 GRANT 权限, 即该用户可以对其他用户赋予权限。

【例 2.4】使用 GRANT 语句创建一个新的用户 testUser, 密码为 testpwd, 并授予用户对所有数据表的 SELECT 和 UPDATE 权限。GRANT 语句及其执行结果如下:

```
MySQL> GRANT SELECT,UPDATE ON *.* TO 'testUser'@'localhost'
-> IDENTIFIED BY 'testpwd';
Query OK, 0 rows affected (0.03 sec)
```

执行结果显示执行成功, 使用 SELECT 语句查询用户 testUser 的权限:

```
MySQL> SELECT Host,User,Select_priv,Update_priv FROM mysql.user where
user='testUser';
+-----+-----+-----+-----+
| Host      | User    | Select_priv | Update_priv |
+-----+-----+-----+-----+
| localhost | testUser | Y           | Y           |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

查询结果显示用户 testUser 被创建成功, 其 SELECT 和 UPDATE 权限字段值均为 'Y'。

提示

User 表中的 user 和 host 字段区分大小写, 在查询的时候要指定正确的用户名或者主机名。

3. 直接操作 MySQL 用户表

通过前面的介绍, 不管是 CREATE USER 或者 GRANT, 在创建新用户时, 实际上都是在 user 表中添加一条新的记录。因此, 可以使用 INSERT 语句向 user 表中直接插入一条记录来创建一个新的用户。使用 INSERT 语句, 必须拥有对 mysql.user 表的 INSERT 权限。使用 INSERT

语句创建新用户的基本语法格式如下：

```
INSERT INTO mysql.user(Host, User, Password, [privilegelist])
VALUES('host', 'username', PASSWORD('password'), privilegevaluelist);
```

Host、User、Password 分别为 user 表中的主机、用户名称和密码字段；privilegelist 表示用户的权限，可以有多个权限；PASSWORD()函数为密码加密函数；privilegevaluelist 为对应的权限的值，只能取'Y'或者'N'。

【例 2.5】使用 INSERT 创建一个新账户，其用户名称为 customer1，主机名称为 localhost，密码为 customer1，INSERT 语句如下：

```
INSERT INTO user (Host,User,Password)
VALUES('localhost','customer1',PASSWORD('customer1'));
```

语句执行结果如下：

```
MySQL> INSERT INTO user (Host,User,Password)
-> VALUES('localhost','customer1',PASSWORD('customer1'));
ERROR 1364 (HY000): Field 'ssl_cipher' doesn't have a default value
```

语句执行失败，查看警告信息如下：

```
MySQL> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message |
+-----+-----+-----+
| Warning | 1364 | Field 'ssl_cipher' doesn't have a default value |
| Warning | 1364 | Field 'x509_issuer' doesn't have a default value |
| Warning | 1364 | Field 'x509_subject' doesn't have a default value |
+-----+-----+-----+
```

因为 ssl_cipher、x509_issuer 和 x509_subject 3 个字段在 user 表定义中没有设置默认值，所以在这里提示错误信息。影响 INSERT 语句的执行，使用 SELECT 语句查看 user 表中的记录：

```
MySQL> SELECT host,user,password FROM user ;
+-----+-----+-----+
| host | user | password |
+-----+-----+-----+
| localhost | root | *0801D10217B06C5A9F32430C1A34E030D41A0257 |
| localhost | jeffrey | *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
| localhost | testUser | *22CBF14EBDE8814586FF12332FA2B6023A7603BB |
```

```

+-----+
3 rows in set (0.00 sec)

```

可以看到添加新用户失败。

2.2.3 删除普通用户

在 MySQL 数据库中，可以使用 DROP USER 语句删除用户，也可以直接通过 DELETE 从 mysql.user 表中删除对应的记录来删除用户。

1. 使用 DROP USER 语句删除用户

DROP USER 语句语法如下：

```
DROP USER user [, user];
```

DROP USER 语句用于删除一个或多个 MySQL 账户。要使用 DROP USER，必须拥有 MySQL 数据库的全局 CREATE USER 权限或 DELETE 权限。使用与 GRANT 或 REVOKE 相同的格式为每个账户命名；例如，“'jeffrey'@'localhost'”账户名称的用户和主机部分与用户表记录的 User 和 Host 列值相对应。

使用 DROP USER，可以删除一个账户和其权限，操作如下：

```

DROP USER 'user'@'localhost';
DROP USER;

```

第 1 条语句可以删除 user 在本地登录权限；第 2 条语句可以删除来自所有授权表的账户权限记录。

【例 2.6】使用 DROP USER 删除账户 “'jeffrey'@'localhost'”，DROP USER 语句如下：

```
DROP USER 'jeffrey'@'localhost';
```

执行过程如下：

```

MySQL> DROP USER 'jeffrey'@'localhost';
Query OK, 0 rows affected (0.00 sec)

```

可以看到语句执行成功，查看执行结果：

```
MySQL> SELECT host,user,password FROM user ;
```


```

+-----+
| host      | user      | password |
+-----+
| localhost | root      | *0801D10217B06C5A9F32430C1A34E030D41A0257 |
| localhost | customer1 | *73DA97747611396FD898E4A7E42B1097B0780646 |

```

```
| localhost | testUser | *22CBF14EBDE8814586FF12332FA2B6023A7603BB |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

user 表中已经没有名称为 jeffrey，主机名为 localhost 的账户，即 “Jeffrey'@'localhost” 的用户账号已经被删除。



提示 DROP USER 不能自动关闭任何打开的用户对话。而且，如果用户有打开的对话，此时取消用户，命令则不会生效，直到用户对话被关闭后才能生效。一旦对话被关闭，用户也被取消，此用户再次试图登录时将会失败。

2. 使用 DELETE 语句删除用户

DELETE 语句基本语法格式如下：

```
DELETE FROM MySQL.user WHERE host='hostname' and user='username'
```

host 和 user 为 user 表中的两个字段，两个字段的组合确定所要删除的账户记录。

【例 2.7】使用 DELETE 删除用户 ‘customer1'@'localhost’，DELETE 语句如下：

```
DELETE FROM MySQL.user WHERE host= 'localhost' and user='customer1';
```

执行结果如下：

```
MySQL> DELETE FROM MySQL.user WHERE host='localhost' and user='customer1';
Query OK, 1 row affected (0.00 sec)
```

可以看到语句执行成功，'customer1'@'localhost'的用户账号已经被删除。读者可以使用 SELECT 语句查询 user 表中的记录，确认删除操作是否成功。

2.2.4 root 用户修改自己的密码

root 用户的安全对于保证 MySQL 的安全非常重要，因为 root 用户拥有很高的权限。修改 root 用户密码的方式有多种，本小节将介绍几种常用的修改 root 用户密码的方法。

1. 使用 mysqladmin 命令在命令行指定新密码

mysqladmin 命令的基本语法格式如下：

```
mysqladmin -u username -h localhost -p password "newpwd"
```

username 为要修改密码的用户名称，在这里指定为 root 用户；参数-h 指需要修改的、对应哪个主机用户的密码，该参数可以不写，默认是 localhost；-p 表示输入当前密码；password 为关键字，后面双引号内的内容 “newpwd” 为新设置的密码。执行完上面的语句，root 用户的密码将被修改为 newpwd。

【例 2.8】使用 `mysqladmin` 将 `root` 用户的密码修改为 “rootpwd”，在 Windows 的命令行窗口中执行命令如下：

```
mysqladmin -u root -p password "rootpwd"
Enter password:
```

按照提示输入 `root` 用户的密码，执行完毕后，新的密码将被设定。`root` 用户登录时将使用新的密码。

2. 修改 mysql 数据库的 user 表

因为所有账户信息都保存在 `user` 表中，因此可以通过直接修改 `user` 表来改变 `root` 用户的密码。`root` 用户登录到 MySQL 服务器后，使用 `UPDATE` 语句修改 `mysql` 数据库的 `user` 表的 `password` 字段，从而修改用户的密码。使用 `UPDATE` 语句修改 `root` 用户密码的语句如下：

```
UPDATE mysql.user set Password=PASSWORD("rootpwd") WHERE User="root" and
Host="localhost";
```

`PASSWORD()` 函数用来加密用户密码。执行 `UPDATE` 语句后，需要执行 `FLUSH PRIVILEGES` 语句重新加载用户权限。

【例 2.9】使用 `UPDATE` 语句将 `root` 用户的密码修改为 “rootpwd2”。

使用 `root` 用户登录到 MySQL 服务器后，执行如下语句：

```
MySQL> UPDATE mysql.user set Password=password("rootpwd2")
-> WHERE User="root" and Host="localhost";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
MySQL> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.11 sec)
```

执行完 `UPDATE` 语句后，`root` 的密码被修改成了 `rootpwd2`。使用 `FLUSH PRIVILEGES` 语句重新加载权限，就可以使用新的密码登录 `root` 用户了。

3. 使用 SET 语句修改 root 用户的密码

`SET PASSWORD` 语句可以用来重新设置其他用户的登录密码或者自己使用的账户的密码。使用 `SET` 语句修改自身密码的语法结构如下：

```
SET PASSWORD=PASSWORD("rootpwd");
```

新密码必须使用 `PASSWORD()` 函数加密。

【例 2.10】使用 `SET` 语句将 `root` 用户的密码修改为 “rootpwd3”。

使用 `root` 用户登录到 MySQL 服务器后，执行如下语句：

```
MySQL> SET PASSWORD=PASSWORD("rootpwd3");
Query OK, 0 rows affected (0.00 sec)
```


SET 语句执行成功，root 用户的密码被成功设置为 rootpwd3。为了使更改生效，需要重新启动 MySQL 或者使用 FLUSH PRIVILEGES; 语句刷新权限，重新加载权限表。

2.2.5 root 用户修改普通用户密码

root 用户拥有很高的权限，不仅可以修改自己的密码，还可以修改其他用户的密码。root 用户登录 MySQL 服务器后，可以通过 SET 语句修改 mysql.user 表，以及 GRANT 语句修改用户的密码。本小节将向读者介绍 root 用户修改普通用户密码的方法。

1. 使用 SET 语句修改普通用户的密码

使用 SET 语句修改其他用户密码的语法格式如下：

```
SET PASSWORD FOR 'user'@'host' = PASSWORD('somepassword');
```

只有 root 可以通过更新 MySQL 数据库的用户来更改其他用户的密码。如果使用普通用户修改，可省略 FOR 子句更改自己的密码：

```
SET PASSWORD = PASSWORD('somepassword');
```

【例 2.11】使用 SET 语句将 testUser 用户的密码修改为 “newpwd”。

使用 root 用户登录到 MySQL 服务器后，执行如下语句：

```
MySQL> SET PASSWORD FOR 'testUser'@'localhost'=PASSWORD("newpwd");
Query OK, 0 rows affected (0.00 sec)
```

SET 语句执行成功，testUser 用户的密码被成功设置为 newpwd。

2. 使用 UPDATE 语句修改普通用户的密码

使用 root 用户登录到 MySQL 服务器后，可以使用 UPDATE 语句修改 MySQL 数据库的 user 表的 password 字段，从而修改普通用户的密码。使用 UPDATE 语句修改用户密码的语法如下：

```
UPDATE MySQL.user SET Password=PASSWORD("pwd")
WHERE User="username" AND Host="hostname";
```

PASSWORD() 函数用来加密用户密码。执行 UPDATE 语句后，需要执行 FLUSH PRIVILEGES 语句重新加载用户权限。

【例 2.12】使用 UPDATE 语句将 testUser 用户的密码修改为 “newpwd2”。

使用 root 用户登录到 MySQL 服务器后，执行如下语句：

```
MySQL> UPDATE MySQL.user SET Password=PASSWORD("newpwd2")
-> WHERE User="testUser" AND Host="localhost";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
MySQL> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.11 sec)
```

执行完 UPDATE 语句后, testUser 的密码被修改成了 newpwd2。使用 FLUSH PRIVILEGES 重新加载权限, 就可以使用新的密码登录 testUser 用户了。

3. 使用 GRANT 语句修改普通用户密码

除了前面介绍的方法, 还可以在全局级别使用 GRANT USAGE 语句(*)指定某个账户的密码而不影响账户当前的权限, 使用 GRANT 语句修改密码, 必须拥有 GRANT 权限。一般情况下最好使用该方法来指定或修改密码:

```
MySQL> GRANT USAGE ON *.* TO 'someuser'@'%' IDENTIFIED BY 'somepassword';
```

【例 2.13】使用 GRANT 语句将 testUser 用户的密码修改为 “newpwd3”。

使用 root 用户登录到 MySQL 服务器后, 执行如下语句:

```
MySQL> GRANT USAGE ON *.* TO 'testUser'@'localhost' IDENTIFIED BY 'newpwd3';
Query OK, 0 rows affected (0.00 sec)
```

执行完 GRANT 语句后, testUser 的密码被修改成了 newpwd3。可以使用新密码登录 MySQL 服务器。

如果使用 GRANT ... IDENTIFIED BY 语句或 mysqladmin password 命令设置密码, 它们均会加密密码。在这种情况下, 不需要使用 PASSWORD()函数。

2.2.6 普通用户修改密码

普通用户登录 MySQL 服务器后, 通过 SET 语句设置自己的密码。

SET 语句修改自己密码的基本语法如下:

```
SET PASSWORD = PASSWORD('newpassword');
```

其中, PASSWORD()函数对密码进行加密, “newpassword”是设置的新密码。

【例 2.14】testUser 用户使用 SET 语句将自身的密码修改为 “newpwd4”:

使用 testUser 用户登录到 MySQL 服务器后, 执行如下语句:

```
MySQL> SET PASSWORD = PASSWORD("newpwd4");
Query OK, 0 rows affected (0.00 sec)
```

SET 语句执行成功, testUser 用户的密码被成功设置为 newpwd4。可以使用新密码登录 MySQL 服务器。

2.2.7 root 用户密码丢失的解决办法

对于 root 用户密码丢失这种特殊情况, MySQL 实现了对应的处理机制。可以通过特殊方法登录到 MySQL 服务器, 然后在 root 用户下重新设置密码。执行步骤如下:

1. 使用--skip-grant-tables 选项启动 MySQL 服务

以 skip-grant-tables 选项启动时, MySQL 服务器将不加载权限判断, 任何用户都能访问数据库。在 Windows 操作系统中, 可以使用 `mysqld` 或 `mysqld-nt` 来启动 MySQL 服务进程。如果 MySQL 的目录已经添加到环境变量中, 可以直接使用 `mysqld`、`mysqld-nt` 命令启动 MySQL 服务。否则需要先在命令行下切换到 MySQL 的 bin 目录。

`mysqld` 命令如下:

```
mysqld --skip-grant-tables
```

`mysqld-nt` 命令如下:

```
mysqld-nt --skip-grant-tables
```

在 Linux 操作系统中, 使用 `mysqld_safe` 来启动 MySQL 服务。也可以使用 `/etc/init.d/mysql` 命令来启动 MySQL 服务。

`mysqld_safe` 命令如下:

```
mysqld_safe --skip-grant-tables user=mysql
```

`/etc/init.d/mysql` 命令如下:

```
/etc/init.d/mysql start-mysqld --skip-grant-tables
```

启动 MySQL 服务后, 就可以使用 root 用户登录了。

2. 使用 root 用户登录, 重新设置密码

在这里使用的平台为 Windows XP, 操作步骤如下:

步骤 01 使用 `net stop mysql` 命令停止 MySQL 服务进程。

```
C:\>net stop MySQL
MySQL 服务正在停止。
MySQL 服务已成功停止。
```

步骤 02 在命令行输入 `mysqld --skip-grant-tables` 选项启动 MySQL 服务。

```
C:\>mysqld --skip-grant-tables
```

提示

命令运行之后, 用户无法输入指令, 此时如果在任务管理器中可以看到名称为 `mysqld` 的进程, 则表示可以使用 root 用户登录 MySQL 了。

步骤 03 打开另外一个命令行窗口, 输入不加密的登录命令。

```
C:\>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.10 MySQL Community Server (GPL)
```

```
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql>
```

登录成功以后, 可以使用 UPDATE 语句或者使用 mysqladmin 命令重新设置 root 密码, 设置密码语句如下:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd') WHERE User='root' and  
Host='localhost';
```

设置 root 密码的方法参见本章第 2.2.4 小节“root 用户修改自己的密码”。

3. 加载权限表

修改密码完成后, 必须使用 FLUSH PRIVILEGES 语句加载权限表。加载权限表后, 新的密码才会生效, 同时 MySQL 服务器开始权限验证。输入语句如下:

```
mysql> FLUSH PRIVILEGES;
```

修改密码完成后, 将输入 mysqld --skip-grant-tables 命令的命令行窗口关闭, 接下来就可以使用新设置的密码登录 MySQL 了。

2.3 权限管理

权限管理主要是对登录到 MySQL 的用户进行权限验证, 所有用户的权限都存储在 MySQL 的权限表中, 不合理的权限规划会给 MySQL 服务器带来安全隐患。数据库管理员要对所有用户的权限进行合理规划管理。MySQL 权限系统的主要功能是证实连接到一台给定主机的用户, 并且赋予该用户在数据库上的 SELECT、INSERT、UPDATE 和 DELETE 权限。本节将为读者介绍 MySQL 权限管理的内容。

2.3.1 MySQL 的各种权限

账户权限信息被存储在 MySQL 数据库的 user、db、host、tables_priv、columns_priv 和 procs_priv 表中。在 MySQL 启动时, 服务器将这些数据库表中权限信息的内容读入内存。

GRANT 和 REVOKE 语句所涉及的权限的名称如表 2-7 所示, 还有在授权表中每个权限的表列名称和每个权限有关的操作对象等。

表 2-7 GRANT 和 REVOKE 语句中可以使用的权限

| 权限 | user 表中对应的列 | 权限的范围 |
|-------------------------|------------------------|------------|
| CREATE | Create_priv | 数据库、表或索引 |
| DROP | Drop_priv | 数据库、表或视图 |
| GRANT OPTION | Grant_priv | 数据库、表或存储过程 |
| REFERENCES | References_priv | 数据库或表 |
| EVENT | Event_priv | 数据库 |
| ALTER | Alter_priv | 数据库 |
| DELETE | Delete_priv | 表 |
| INDEX | Index_priv | 表 |
| INSERT | Insert_priv | 表 |
| SELECT | Select_priv | 表或列 |
| UPDATE | Update_priv | 表或列 |
| CREATE TEMPORARY TABLES | Create tmp table_priv | 表 |
| LOCK TABLES | Lock tables_priv | 表 |
| TRIGGER | Trigger_priv | 表 |
| CREATE VIEW | Create view_priv | 视图 |
| SHOW VIEW | Show view_priv | 视图 |
| ALTER ROUTINE | Alter routine_priv | 存储过程和函数 |
| CREATE ROUTINE | Create routine_priv | 存储过程和函数 |
| EXECUTE | Execute_priv | 存储过程和函数 |
| FILE | File_priv | 访问服务器上的文件 |
| CREATE TABLESPACE | Create tablespace_priv | 服务器管理 |
| CREATE USER | Create user_priv | 服务器管理 |
| PROCESS | Process_priv | 存储过程和函数 |
| RELOAD | Reload_priv | 访问服务器上的文件 |
| REPLICATION CLIENT | Repl client_priv | 服务器管理 |
| REPLICATION SLAVE | Repl slave_priv | 服务器管理 |
| SHOW DATABASES | Show db_priv | 服务器管理 |
| SHUTDOWN | Shutdown_priv | 服务器管理 |
| SUPER | Super_priv | 服务器管理 |

- (1) CREATE 和 DROP 权限，可以创建新数据库和表，或删除（移掉）已有数据库和表。如果将 MySQL 数据库中的 DROP 权限授予某用户，用户可以删掉 MySQL 访问权限保存的数据库。
- (2) SELECT、INSERT、UPDATE 和 DELETE 权限允许在一个数据库现有的表上实施操作。
- (3) SELECT 权限只有在它们真正从一个表中检索时才被用到。
- (4) INDEX 权限允许创建或删除索引，INDEX 适用于已有表。如果具有某个表的 CREATE 权限，可以在 CREATE TABLE 语句中包括索引定义。

- (5) ALTER 权限，可以使用 ALTER TABLE 来更改表的结构和重新命名表。
- (6) CREATE ROUTINE 权限用来创建保存的程序（函数和程序），ALTER ROUTINE 权限用来更改和删除保存的程序，EXECUTE 权限用来执行保存的程序。
- (7) GRANT 权限允许授权给其他用户。可用于数据库、表和保存的程序。
- (8) FILE 权限给予用户使用 LOAD DATA INFILE 和 SELECT ... INTO OUTFILE 语句读或写服务器上的文件，任何被授予 FILE 权限的用户都能读或写 MySQL 服务器上的任何文件。（说明用户可以读任何数据库目录下的文件，因为服务器可以访问这些文件）。FILE 权限允许用户在 MySQL 服务器具有写权限的目录下创建新文件，但不能覆盖已有文件。

其余的权限用于管理性操作，它使用 mysqladmin 程序或 SQL 语句实施。表 2-8 显示每个权限允许执行的 mysqladmin 命令。

表 2-8 不同权限下可以使用的 mysqladmin 命令

| 权限 | 权限拥有者允许执行的命令 |
|----------|------------------------------------------------------------------------------------------------|
| RELOAD | flush-hosts,flush-logs,flush-privileges,flush-status,flush-tables,flush-threads,refresh,reload |
| SHUTDOWN | shutdown |
| PROCESS | processlist |
| SUPER | kill |

- (1) reload 命令告诉服务器将授权表重新读入内存；flush-privileges 是 reload 的同义词；refresh 命令清空所有表并关闭/打开记录文件；其他 flush-xxx 命令执行类似 refresh 的功能，但是范围更有限，并且在某些情况下可能更好用。例如，如果只是想清空记录文件，flush-logs 是比 refresh 更好的选择。
- (2) shutdown 命令关掉服务器。只能从 mysqladmin 发出命令。
- (3) processlist 命令显示在服务器内执行的线程的信息（即其他账户相关的客户端执行的语句）。kill 命令杀死服务器线程。用户总是能显示或杀死自己的线程，但是需要 PROCESS 权限来显示或杀死其他用户和 SUPER 权限启动的线程。
- (4) kill 命令能用来终止其他用户或更改服务器的操作方式。

总的来说，只授予权限给需要他们的那些用户。

2.3.2 授权

授权就是为某个用户授予权限。合理的授权可以保证数据库的安全。MySQL 中可以使用 GRANT 语句为用户授予权限。

授予的权限可以分为多个层级。

1. 全局层级

全局权限适用于一个给定服务器中的所有数据库。这些权限存储在 mysql.user 表中。GRANT ALL ON *.*和 REVOKE ALL ON *.*只授予和撤销全局权限。

2. 数据库层级

数据库权限适用于一个给定数据库中的所有目标。这些权限存储在 `mysql.db` 和 `mysql.host` 表中。GRANT ALL ON `db_name`. 和 REVOKE ALL ON `db_name.*` 只授予和撤销数据库权限。

3. 表层级

表权限适用于一个给定表中的所有列。这些权限存储在 `mysql.tables_priv` 表中。GRANT ALL ON `db_name.tbl_name` 和 REVOKE ALL ON `db_name.tbl_name` 只授予和撤销表权限。

4. 列层级

列权限适用于一个给定表中的单一列。这些权限存储在 `mysql.columns_priv` 表中。当使用 REVOKE 时，必须指定与被授权列相同的列。

5. 子程序层级

CREATE ROUTINE、ALTER ROUTINE、EXECUTE 和 GRANT 权限适用于已存储的子程序。这些权限可以被授予为全局层级和数据库层级。而且，除了 CREATE ROUTINE 外，这些权限可以被授予子程序层级，并存储在 `mysql.procs_priv` 表中。

在 MySQL 中，必须是拥有 GRANT 权限的用户才可以执行 GRANT 语句。

要使用 GRANT 或 REVOKE，必须拥有 GRANT OPTION 权限，并且必须用于正在授予或撤销的权限。GRANT 的语法如下：

```
GRANT priv_type [(columns)] [, priv_type [(columns)]] ...
ON [object_type] table1, table2, ..., tablenn
TO user [IDENTIFIED BY [PASSWORD] 'password']
[, user [IDENTIFIED BY [PASSWORD] 'password']] ...
[WITH GRANT OPTION]

object_type = TABLE | FUNCTION | PROCEDURE

GRANT OPTION 取值：
| MAX_QUERIES_PER_HOUR count
| MAX_UPDATES_PER_HOUR count
| MAX_CONNECTIONS_PER_HOUR count
| MAX_USER_CONNECTIONS count
```

其中，`priv_type` 参数表示权限类型；`columns` 参数表示权限作用于哪些列上，不指定该参数，表示作用于整个表；`table1, table2, ..., tablenn` 表示授予权限的列所在的表；`object_type` 指定授权作用的对象类型，包括 TABLE（表）、FUNCTION（函数）和 PROCEDURE（存储过程），当从旧版本的 MySQL 升级时，要使用 `object_type` 子句，必须升级授权表；`user` 参数表示用户账户，由用户名和主机名构成，形式是 “`'username'@'hostname'`”；IDENTIFIED BY 参数用于设置密码。

WITH 关键字后可以跟一个或多个 GRANT OPTION。GRANT OPTION 的取值有 5 个，意义如下：

- GRANT OPTION, 将自己的权限赋予其他用户。
- MAX_QUERIES_PER_HOUR count, 设置每个小时可以执行 count 次查询。
- MAX_UPDATES_PER_HOUR count, 设置每小时可以执行 count 次更新。
- MAX_CONNECTIONS_PER_HOUR count, 设置每小时可以建立 count 个连接。
- MAX_USER_CONNECTIONS count, 设置单个用户可以同时建立 count 个连接。

【例 2.15】使用 GRANT 语句创建一个新的用户 grantUser, 密码为“grantpwd”。用户 grantUser 对所有的数据有查询、插入权限, 并授予 GRANT 权限。GRANT 语句及其执行结果如下:

```
MySQL> GRANT SELECT,INSERT ON *.* TO 'grantUser'@'localhost'
-> IDENTIFIED BY 'grantpwd'
-> WITH GRANT OPTION;
Query OK, 0 rows affected (0.03 sec)
```

结果显示执行成功, 使用 SELECT 语句查询用户 grantUser 的权限:

```
MySQL> SELECT Host,User,Select_priv,Insert_priv, Grant_priv FROM mysql.user
where user='grantUser';
+-----+-----+-----+-----+-----+
| Host      | User      | Select_priv | Insert_priv | Grant_priv |
+-----+-----+-----+-----+-----+
| localhost | grantUser | Y           | Y           | Y           |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

查询结果显示用户 grantUser 2 被创建成功, 并被赋予 SELECT、INSERT 和 GRANT 权限, 其相应字段值均为 Y。

被授予 GRANT 权限的用户可以登录 MySQL 并创建其他用户账户, 在这里为名称是 grantUser 的用户。读者可以使用 grantUser 登录, 并按照【例 2.4】中的过程创建并授权其他账户。

2.3.3 收回权限

收回权限就是取消已经赋予用户的某些权限。收回用户不必要的权限可以在一定程度上保证系统的安全性。MySQL 中使用 REVOKE 语句取消用户的某些权限。使用 REVOKE 收回权限之后, 用户账户的记录将从 db、host、tables_priv 和 columns_priv 表中删除, 但是用户账号记录仍然在 user 表中保存(删除 user 表中的账户记录, 使用 DROP USER 语句, 在 2.2.3 节已经介绍)。

在将用户账户从 user 表删除之前, 应该收回相应用户的所有权限, REVOKE 语句有两种语法格式, 第一种语法是收回所有用户的所有权限, 此语法用于取消对于已命名的用户的所有全局层级、数据库层级、表层级和列层级的权限, 其语法如下:


```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM 'user'@'host' [, 'user'@'host' ...]
```

REVOKE 语句必须和 FROM 语句一起使用，FROM 语句指明需要收回权限的账户。

另一种为长格式的 REVOKE 语句，基本语法如下：

```
REVOKE priv_type [(columns)] [, priv_type [(columns)]] ...
ON table1, table2,..., tablen
FROM 'user'@'host'[, 'user'@ 'host' ...]
```

该语法收回指定的权限。其中，priv_type 参数表示权限类型；columns 参数表示权限作用于哪些列上，如果不指定该参数，表示作用于整个表；table1,table2,...,tablen 表示从哪个表中收回权限；'user'@'host'参数表示用户账户，由用户名和主机名构成。

要使用 REVOKE 语句，必须拥有 mysql 数据库的全局 CREATE USER 权限或 UPDATE 权限。

【例 2.16】使用 REVOKE 语句取消用户 testUser 的更新权限。REVOKE 语句及其执行结果如下：

```
MySQL> REVOKE UPDATE ON *.* FROM 'testUser'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

执行结果显示执行成功，使用 SELECT 语句查询用户 testUser 的权限：

```
MySQL> SELECT Host,User,Select_priv,Update_priv,Grant_priv FROM MySQL.user
where user='testUser';
```

| Host | User | Select_priv | Update_priv | Grant_priv |
|-----------|----------|-------------|-------------|------------|
| localhost | testUser | Y | N | Y |

```
1 row in set (0.00 sec)
```

查询结果显示用户 testUser 的 Update_priv 字段值为“N”，UPDATE 权限已经被收回。

提示

当从旧版本的 MySQL 升级时，如果要使用 EXECUTE、CREATE VIEW、SHOW VIEW、CREATE USER、CREATE ROUTINE 和 ALTER ROUTINE 权限，必须首先升级授权表。

2.3.4 查看权限

SHOW GRANTS 语句可以显示指定用户的权限信息，使用 SHOW GRANTS 查看账户信息的基本语法格式如下：

```
SHOW GRANTS FOR 'user'@ 'host' ;
```

其中, user 表示登录用户的名称, host 表示登录的主机名称或者 IP 地址。在使用该语句时,要确保指定的用户名和主机名都要用单引号括起来,并使用 '@' 符号,将两个名字分隔开。

【例 2.17】使用 SHOW GRANTS 语句查询用户 testUser 的权限信息。SHOW GRANTS 语句及其执行结果如下:

```
MySQL> SHOW GRANTS FOR 'testUser'@'localhost';
```

```
+-----+
| Grants for testUser@localhost |
+-----+
| GRANT SELECT ON *.* TO 'testUser'@'localhost' IDENTIFIED BY PASSWORD |
| '*53835E70E1FC57BE1A455169C761A8778D307C81' WITH GRANT OPTION |
+-----+
1 row in set (0.00 sec)
```

返回结果的第 1 行显示了 user 表中的账户信息;接下来的行以 GRANT SELECT ON 关键字开头,表示用户被授予了 SELECT 权限; *.* 表示 SELECT 权限作用于所有数据库的所有数据表; IDENTIFIED BY PASSWORD 关键字后面为用户加密后的密码。

在这里,只是定义了个别的用户权限,GRANT 可以显示更加详细的权限信息,包括全局级的和非全局级的权限,如果表层级或者列层级的权限被授予用户的话,它们也能在结果中显示出来。

在前面创建用户时,查看新建的账户时使用 SELECT 语句,也可以通过 SELECT 语句查看 user 表中的各个权限字段以确定用户的权限信息,其基本语法格式如下:

```
SELECT privileges_list FROM user WHERE user='username', host='hostname';
```

其中, privileges_list 为想要查看的权限字段,可以为 Select_priv、Insert_priv 等。读者可以根据需要选择要查询的字段。

2.4 访问控制

正常情况下,并不希望每个用户都可以执行所有的数据库操作。当 MySQL 允许一个用户执行各种操作时,它将首先核实该用户向 MySQL 服务器发送的连接请求,然后确认用户的操作请求是否被允许。本小节将向读者介绍 MySQL 中的访问控制过程。MySQL 的访问控制分为两个阶段:连接核实阶段和请求核实阶段。

2.4.1 连接核实阶段

当连接 MySQL 服务器时,服务器基于用户的身份以及用户是否能通过正确的密码身份验证来接受或拒绝连接,即客户端用户连接请求中会提供用户名称、主机地址名和密码,MySQL

使用 user 表中的 3 个字段（Host、User 和 Password）执行身份检查，服务器只有在 user 表记录的 Host 和 User 字段匹配客户端主机名和用户名，并且提供正确的密码时才接受连接。如果连接核实没有通过，服务器完全拒绝访问；否则，服务器接受连接，然后进入阶段 2 等待用户请求。

2.4.2 请求核实阶段

建立了连接之后，服务器进入访问控制的阶段 2。对在此连接上进来的每个请求，服务器检查用户要执行的操作，然后检查是否有足够的权限来执行它。这正是在授权表中的权限列发挥作用的地方。这些权限可以来自 user、db、host、tables_priv 或 columns_priv 表。

确认权限时，MySQL 首先检查 user 表，如果指定的权限没有在 user 表中被授权，MySQL 将检查 db 表，db 表是下一安全层级，其中的权限限定于数据库层级，在该层级的 SELECT 权限允许用户查看指定数据库的所有表中的数据；如果在该层级没有找到限定的权限，则 MySQL 继续检查 tables_priv 表以及 columns_priv 表，如果所有权限表都检查完毕，但还是没有找到允许的权限操作，MySQL 将返回错误信息，用户请求的操作不能执行，操作失败。

请求核实的过程如图 2-1 所示。

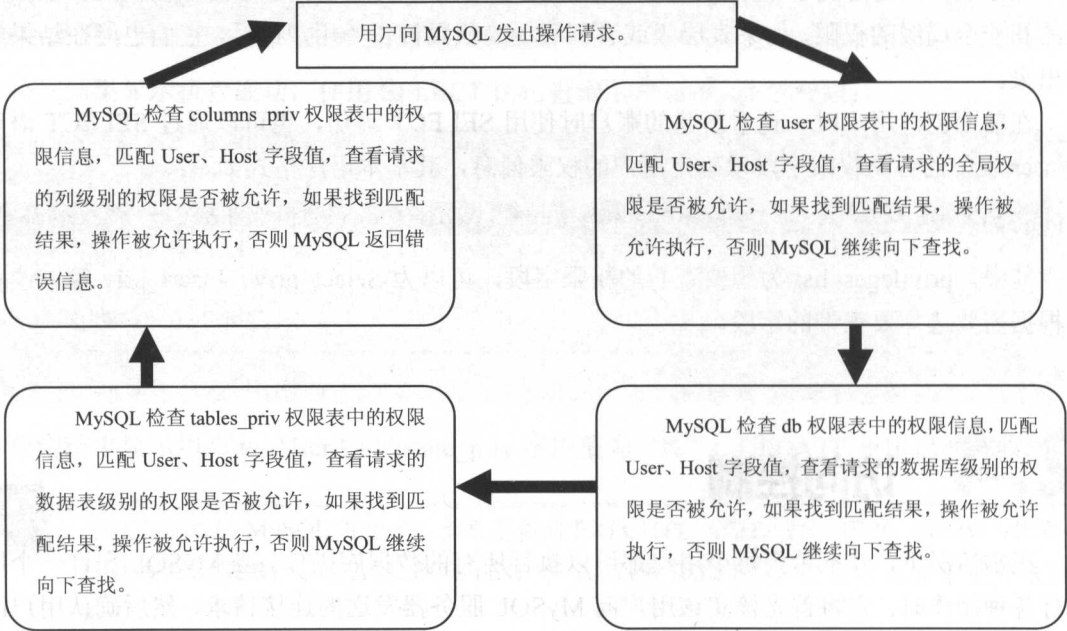


图 2-1 MySQL 请求核实过程

值得注意的是，MySQL 身份验证是通过身份和 IP 地址联合进行认证的。例如 MySQL 默认账户 root@localhost 表示 root 用户只能通过本地才能访问数据库。对于同一个用户如果来自不同的 IP 地址，MySQL 则视为不同的用户。

提示

MySQL 通过向下层级的顺序检查权限表（从 `user` 表到 `columns_priv` 表），但并不是所有的权限都要执行该过程。例如，一个用户登录到 MySQL 服务器之后只执行对 MySQL 的管理操作，此时，只涉及管理权限，因此 MySQL 只检查 `user` 表。另外，如果请求的权限操作不被允许，MySQL 也不会继续检查下一层级的表。

2.5 MySQL 的安全问题

如果数据库的安全性不高，轻则数据被窃取，重则数据被破坏。本章节主要通过操作系统和数据库两个层面对 MySQL 的安全问题进行讨论。

2.5.1 操作系统相关的安全问题

数据库的安全性固然重要，但是如果操作系统本身存在安全问题的话，数据库的安全性就没有一点保障了。本节介绍一些常见的操作系统存在的安全问题，这些问题出现在 MySQL 的安装和启动过程中，希望读者对操作系统的安全性能重视起来。

1. 尽量避免以 root 用户运行 MySQL 服务器

在 Linux 操作系统下，MySQL 数据库目录的属主为 `root`，数据目录的属主为 `mysql` 用户。`mysql` 用户专门负责 MySQL 数据库的启动和关闭，这样做可以防止任何具有 `FILE` 权限的用户能够访问 `root` 用户创建的文件。如果任何具有 `FILE` 权限的用户能够读写 `root` 文件，这样操作系统就存在一定的安全隐患。此时应该使用普通非特权用户运行 `mysqld`，该账户只用来管理和运行 MySQL 服务器。

要想使普通非特权用户启动 `mysqld`，可以修改 `/etc/my.cnf` 文件或者数据库目录的 `my.cnf` 文件，MySQL 数据库可以在 `my.cnf` 中配置用户名，如下所示：

```
[mysqld]
user = mysql
```

此时服务器启动的时候会根据 `my.cnf` 配置文件中指定的用户来启动，值得注意的是，无论是通过手动启动或通过 `mysql_safe` 或 `mysql.server` 启动，都应该确保使用 `mysql` 的身份，也可以在启动数据库时加上 `user` 参数，命令如下：

```
#/user/local/mysql/bin/mysqld_safe -user=mysql &
```

Linux 用户启动 `mysqld` 时，为了确保 `mysqld` 运行，只使用对数据目录具有读或者写权限的 Linux 用户来启动。绝对不能使用 Linux 系统的 `root` 用户运行 MySQL 服务器。

2. 尽量关闭不需要的服务

尽量关闭操作系统中不需要的服务,这样可以降低操作系统的负担,调高主机的运行性能,从安全方面降低了潜在的安全隐患,可以使用网络扫描工具扫描主机的端口,检测哪些端口是正在监听的,并去掉不必要的端口。通常操作系统端口数的范围为 0~65535,其中 0~1023 的端口最常用,例如,常见的 FTP、SSH、HTTP、NNTP 等服务。通过端口扫描工具,可以帮助管理员修正操作系统的漏洞。

下面介绍五款优秀的端口扫描工具,供有兴趣的读者参考。

(1) Nmap 工具

Nmap (又称为扫描之王)是一款网络扫描软件,用来扫描网络上主机开放的网络连接端,判断主机属于哪种操作系统,操作系统开启了哪些端口,还可以高频率扫描 IP 地址。该工具 是网络管理员比较常用的软件之一,通常被用来查看电脑的网络情况。

(2) Unicornscan 工具

Unicornscan 是一款通过尝试连接用户系统分布式 TCP/IP 堆栈获得信息和关联关系的端口扫描器,它的主要功能包括异步无状态 TCP 扫描、异步无状态 TCP 标志捕获、通过分析反馈信息获取主动/被动远程操作系统、应用程序和组件信息。

(3) Zenmap 工具

端口扫描器 Zenmap 是一款开放源代码的网络探测和安全审核的工具。它的设计目标是快速地扫描大型网络,当然用它扫描单个主机也没有问题。Zenmap 以新颖的方式使用原始 IP 报文来发现网络上有哪些主机、提供什么服务(应用程序名和版本)、运行在什么操作系统上、使用什么类型的报文过滤器和防火墙等等。端口扫描器 Zenmap 通常用于安全审核,许多系统管理员和网络管理员也用它来做一些日常的工作,比如查看整个网络的信息,管理服务升级计划,以及监视主机和服务的运行。

(4) Nast

Nast 工具是一个网络包嗅探和局域网分析工具,基于 libnet 和 libpcap 开发,被称为网络工具中的瑞士军刀。它是一个简单但非常实用的网络工具,通过使用 TCP 或 UDP 协议的网络连接去读写数据。同时它也是一个功能强大的网络测试工具,能够建立所需的几乎任何类型的网络连接。

(5) Knocker 工具

Knocker 是一款简单的 TCP 端口扫描工具,采用 C 语言编写,用于分析主机上运行的服务。

2.5.2 数据库相关的安全问题

数据库的安全性主要是由内部安全和操作系统两部分决定的,对系统管理员来说,首先要保证系统本身安全。本节介绍一些常见的数据库安全问题,很多问题都是由于数据库账号管理

不当造成的。

1. 修改 root 用户口令和删除匿名账号

MySQL 安装后的 root 用户是空密码，为了安全起见最好修改密码，可以使用 MySQL 自带的命令工具 mysqladmin 修改，也可以修改 mysql 下的 user 表字段。

有些版本的 MySQL 安装完毕后，MySQL 初始化后自动生成匿名账号和 test 库，进行安装测试，这对数据库的安全构成威胁，有必要全部删除，最后只保留单个 root 账户即可，当然以后根据需要增加用户和数据库。

首先查看存在的账户名称，操作如下：

```
mysql> use mysql;
Database changed
mysql> select host,user,password from user where user = ' ';
+-----+-----+-----+
| host          | user | password |
+-----+-----+-----+
| localhost     |      |          |
| localhost.localdomain |      |          |
+-----+-----+-----+
2 rows in set (0.00 sec)
```



```
mysql> select host,db,select_priv,update_priv,delete_priv from db where user
= ' ';
+-----+-----+-----+-----+-----+-----+
| host | db      | select_priv | update_priv | delete_priv | ...      |
+-----+-----+-----+-----+-----+-----+
| %    | test   | Y           | Y           | Y           | Y           |
| %    | test\_% | Y           | Y           | Y           | Y           |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

普通用户只需要执行 mysql 命令就可以登录 MySQL 数据库，这个时候默认使用匿名账号，该账户在 test 数据库中 can 执行各种操作，拥有大部分权限。例如，该用户可以批量增加数据，占用大量磁盘空间，这样将给系统造成一定的安全隐患。

下面使用匿名账号登录数据库并做一些操作，如下所示：

```
[root@localhost ~]# mysql -u ' '
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test;
```

```
Database changed
mysql> create table t(id int);
Query OK, 0 rows affected (0.05 sec)

mysql> insert into t values(1);
Query OK, 1 row affected (0.00 sec)
```

建议删除此匿名账户，操作命令如下：

```
[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> drop user ' ' @localhost;
Query OK, 0 rows affected (0.00 sec)
```

2. 设置安全的密码

在设置数据库账户密码时，建议使用 6 位以上字母、数字、下划线和一些特殊的字符构成的安全密码。在 MySQL 中，使用密码通常有以下几种方法。

(1) 直接将密码写在命令行中，如下所示：

```
[root@localhost ~]#mysql -u hakerivan -p feifei123 -h 192.168.0.105
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

(2) 使用交互的方式输入账户密码，如下所示：

```
[root@localhost ~]#mysql -u hakerivan -p -h 192.168.0.105
Enter password: ***
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 5.6.10-log MySQL Community Server (GPL)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

(3) 将用户名和密码写在配置文件中，数据库连接的时候会自动读取，MySQL 数据库可以在 `my.cnf` 中配置用户名和密码信息，如下所示：

```
[client]
user = username
```



```
password= your_password
port      = 3306
socket    = /tmp/mysql.sock
```

3. 禁止远程连接数据库

MySQL 启动默认的端口 3306 是打开的, 此时打开了 mysqld 的网络监听, 允许用户远程通过账号密码连接本地数据库, MySQL 数据库默认是允许远程用户连接服务器的。如果要禁止用户通过远程连接数据, 此时需要启动 skip-networking, 将 my.cnf 的 skip-networking 注释去掉即可。

```
#vi /etc/my.cnf
将#skip-networking 注释去掉
```

如果确实需要提供远程用户访问数据库, 此时应该考虑修改默认的监听端口, 同时添加防火墙规则, 只允许信任网络的 mysql 监听端口的数据通过。

```
[root@localhost ~]# mysqladmin -u root -p shutdown
[root@localhost ~]# mysqld_safe -user=mysql &
```

4. 不要把 FILE、PROCESS 或者 SUPER 权限授予管理员以外的账户

FILE 权限主要用于将数据库的信息通过 SELECT... INTO OUTFILE ... 写到服务器上有写权限的目录下, 然后作为文本格式保存。FILE 权限可以将服务器主机上有读权限的文本文件通过 LOAD DATA INFILE...命令写入到数据库表中。如果这些表中存放了很重要的信息, 将对系统造成很大的安全隐患。

下面以案例进行讲解, 具体的操作如下:

步骤 01 连接数据库并创建表 t_pass, 命令如下所示:

```
[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use mysql;
Database changed
mysql> create table t_pass(name varchar(200));
Query OK, 0 rows affected (0.00 sec)
```

步骤 02 将/etc/passwd 文件加载到表 t_pass 中, 如下所示:

```
mysql> load data infile '/etc/passwd' into table t_pass;
Query OK, 40 rows affected (0.00 sec)
```


Records: 40 Deleted: 0 Skipped: 0 Warnings: 0

mysql> select * from t_pass;

| name | |
|-----------------------------------------------|--|
| root:x:0:0:root:/root:/bin/bash | |
| bin:x:1:1:bin:/bin:/sbin/nologin | |
| daemon:x:2:2:daemon:/sbin:/sbin/nologin | |
| adm:x:3:4:adm:/var/adm:/sbin/nologin | |
| lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin | |
| sync:x:5:0:sync:/sbin:/bin/sync | |
| ... | |
| hsqldb:x:96:96::/var/lib/hsqldb:/sbin/nologin | |
| ntp:x:38:38::/etc/ntp:/sbin/nologin | |
| squid:x:23:23::/var/spool/squid:/sbin/nologin | |
| mysql:x:500:500::/home/mysql:/bin/bash | |

40 rows in set (0.00 sec)

此时系统的一些重要信息写入到表中，造成了系统的安全隐患。

PROCESS 权限主要是用来限制命令“show processlist”，查看所有用户执行查询的文本，包括制订或改变密码的查询。在默认状况下，每个用户都可以执行“show processlist”命令，因此，PROCESS 权限存在一定的安全风险。

下面以案例进行讲解，具体的操作如下：

步骤 01 将 PROCESS 权限授权给普通用户，命令如下所示：

```
[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show processlist;
```

| Id | User | Host | db | Command | Time | State | Info |
|----|------|-----------|-------|---------|------|-------|------------------|
| 1 | root | localhost | mysql | Sleep | 1864 | | NULL |
| 2 | root | localhost | NULL | Query | 0 | NULL | show processlist |

```

+---+-----+-----+-----+-----+-----+-----+-----+
-+
2 rows in set (0.00 sec)

```

```

mysql> grant process on *.* to kobe@localhost;
Query OK, 0 rows affected (0.00 sec)

```

步骤 02 锁定表 user，为了方便其他用户查询进程的内容，如下所示：

```

mysql> lock table user read;
Query OK, 0 rows affected (0.00 sec)

```

步骤 03 此时开启一个 session，root 用户修改密码，如下所示：

```

[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> set password=password('123');

```

此时进程发生了阻塞状态。

步骤 04 开启新的 session，使用账户 kobe 登录，执行“show processlist”命令，此时会清楚地看到 root 用户修改密码的操作，如下所示：

```

[root@localhost ~]# mysql -u kobe -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show processlist;
+---+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host      | db    | Command | Info                                |
+---+-----+-----+-----+-----+-----+-----+-----+
| 1 | root | localhost | mysql | Sleep   | NULL                                |
| 2 | root | localhost | mysql | Sleep   | NULL                                |
| 3 | root | localhost | NULL  | Query   | set password=password('123')      |
| 5 | kobe | localhost | NULL  | Query   | show processlist                   |
+---+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

由此可以发现 PROCESS 权限存在一定的安全隐患,对于普通用户,不应该赋予 PROCESS 权限。

对于 SUPER 权限可以用来执行 kill 命令,关闭掉其他的进程,普通用户如果具有 SUPER 权限,便可以关闭掉任何用户的进程。

下面以案例进行讲解,具体的操作如下:

步骤 01 首先查看用户,并测试用户是否具有 SUPER 权限,如下所示:

```
[root@localhost ~]# mysql -u nba -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 37
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show processlist;
+-----+-----+-----+-----+-----+-----+-----+-----+
| Id | User | Host      | db   | Command | Time | State | Info          |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 8 | kobe | localhost | mysql | Sleep   | 870 |      | NULL          |
| 9 | james | localhost | NULL | Sleep   | 1076 |      | NULL          |
| 14 | bosh | localhost | NULL | Sleep   | 844 |      | NULL          |
| 36 | root | localhost | NULL | Sleep   | 32 |      | NULL          |
| 37 | nba  | localhost | NULL | Query   | 0 | NULL | show processlist |
+-----+-----+-----+-----+-----+-----+-----+-----+

5 rows in set (0.00 sec)

mysql> kill 14;
ERROR 1095 (HY000): You are not owner of thread 14
mysql>
```

可见此时普通用户 nba@localhost 没有 kill 权限。

步骤 02 登录 root 账户,并授予 nba@localhost 用户 KILL 权限,如下所示:

```
[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 39
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```



```
mysql> grant super on *.* to nba@localhost;
Query OK, 0 rows affected (0.00 sec)
```

步骤 03 查看 nba@localhost 用户的权限，如下所示：

```
mysql> show grants for nba@localhost;
+-----+
| Grants for nba@localhost |
+-----+
| GRANT PROCESS, SHOW DATABASES, SUPER ON *.* TO 'nba'@'localhost'|
+-----+
1 row in set (0.00 sec)
```

5. 限制单个用户连接次数

某个数据库用户多次远程连接数据库，将会导致数据库性能的下降和影响其他用户的操作，有必要对其进行限制，设置/etc/my.cnf 文件中[mysqld]选项中的 max_user_connections 参数，可以用来限制单个用户允许连接的次数。

```
[root@localhost ~]# vi /etc/my.cnf
[mysqld]
port      = 3306
socket     = /tmp/mysql.sock
max_user_connections = 2
...
```

6. REVOKE 命令的漏洞

当某个普通用户多次授权后，由于各种原因，需要取消该用户的所有权限，此时，REVOKE 命令存在一定的漏洞，如下面的例子：

步骤 01 登录 root 用户，命令如下所示：

```
[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

步骤 02 创建一个普通用户 tt@localhost，命令如下所示：

```
mysql> create user tt@localhost identified by '123';
Query OK, 0 rows affected (0.01 sec)
```

步骤 03 对用户 tt 连续两次赋权，命令如下所示：


```
mysql> grant select,insert on test.* to tt@localhost;
Query OK, 0 rows affected (0.00 sec)
mysql> grant all privileges on *.* to tt@localhost;
Query OK, 0 rows affected (0.00 sec)
```

步骤 04 查看用户 tt@localhost 的权限, 命令如下所示:

```
mysql> show grants for tt@localhost;
+-----+
| Grants for tt@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'tt'@'localhost' |
| GRANT SELECT, INSERT ON `test`.* TO 'tt'@'localhost' |
+-----+
2 rows in set (0.04 sec)
```

步骤 05 回收用户 tt@localhost 的所有权限, 然后再次查看 tt@localhost 的权限, 如下所示:

```
mysql> revoke all privileges on *.* from tt@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> show grants for tt@localhost;
+-----+
| Grants for tt@localhost |
+-----+
| GRANT USAGE ON *.* TO 'tt'@'localhost' |
| GRANT SELECT, INSERT ON `test`.* TO 'tt'@'localhost' |
+-----+
2 rows in set (0.00 sec)
```

步骤 06 使用用户 tt@localhost 登录 MySQL 数据库, 测试下该用户是否拥有其他操作权限, 命令如下:

```
[root@localhost ~]# mysql -u tt -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 41
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
```

```

+-----+
| t1      |
| t2      |
+-----+
2 rows in set (0.00 sec)

mysql> insert into t1 values('test');
Query OK, 1 row affected (0.03 sec)

```

由上面的例子可以看出, REVOKE 权限是存在一定的安全隐患的, 在回收权限的时候, 使用 REVOKE ALL PRIVILEGES 时最好看下该用户是否还存在其他的操作权限, 数据库回收权限应该使用 REVOKE 命令单独进行权限回收。

2.6 使用 SSL 安全连接

SSL (Secure Socket Layer) 是 Netscape 公司所研发, 利用 Encryption 技术可确保数据在网络传输过程中不被截取及窃听。该技术目前广泛应用于浏览器与服务器之间的身份认证和加密数据传输。

SSL 协议提供的服务主要包括如下:

- 认证用户和服务器, 确保数据发送到正确的客户机和服务器。
- 加密数据以防止被窃取。
- 维护数据的完整性, 确保数据在传输的过程中不被改变。

下面的例子详细介绍 MySQL 服务器使用 SSL 安全传输的过程。

步骤 01 下载 openssl 工具, 然后安装, 命令如下:

```

[root@localhost ~]# tar zxvf openssl-0.9.6.tar.gz
[root@localhost ~]# cd openssl-0.9.6
[root@localhost openssl-0.9.6]# ./config
[root@localhost openssl-0.9.6]# make && make install

```

步骤 02 查看 MySQL 服务器中是否支持 SSL 功能, 命令如下:

```

[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```

```
mysql> show variables like '%ssl%';
```

```
+-----+-----+
| Variable_name | Value      |
+-----+-----+
| have_openssl  | DISABLED  |
| have_ssl      | DISABLED  |
| ssl_ca        |           |
| ssl_capath    |           |
| ssl_cert      |           |
| ssl_cipher    |           |
| ssl_key       |           |
+-----+-----+
```

```
7 rows in set (0.04 sec)
```

```
[root@localhost ~]# mysqladmin -u root -p shutdown
```

```
Enter password:
```

```
[root@localhost ~]#
```

步骤 03 修改 MySQL 服务器配置文件 my.cnf，使其开启 SSL 功能。

```
[root@localhost ~]# vi /etc/my.cnf
```

步骤 04 编辑全局配置文件，使其[mysqld]后面加上“ssl”，然后重启 MySQL 服务器。

```
# The MySQL server
[mysqld]
port      = 3306
socket     = /tmp/mysql.sock
skip-locking
key_buffer = 16M
max_allowed_packet = 1M
table_cache = 64
sort_buffer_size = 512K
net_buffer_length = 8K
read_buffer_size = 256K
read_rnd_buffer_size = 512K
myisam_sort_buffer_size = 8M
ssl
```

```
[root@localhost ~]# mysql -u root -p
```

```
Enter password:
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 1
```

```
Server version: 5.6.10-log MySQL Community Server (GPL)
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```



```
mysql> show variables like '%ssl%';
```

```
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_openssl  | YES   |
| have_ssl      | YES   |
| ssl_ca        |       |
| ssl_capath    |       |
| ssl_cert      |       |
| ssl_cipher    |       |
| ssl_key       |       |
+-----+-----+
7 rows in set (0.00 sec)
```

步骤 05 使用 openssl 命令生成 CA 的密钥对。

```
[root@localhost ~]# mkdir -p /etc/mysql/newcerts
[root@localhost ~]# cd /etc/mysql/newcerts
[root@localhost newcerts]# openssl genrsa 2048 > ca-key.pem
Generating RSA private key, 2048 bit long modulus
.....+++
.....
+++
e is 65537 (0x10001)

[root@localhost newcerts]# openssl req -new -x509 -nodes -days 1000 -key
ca-key.pem > ca-cert.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:zg
State or Province Name (full name) [Berkshire]:hubei
Locality Name (eg, city) [Newbury]:xf
Organization Name (eg, company) [My Company Ltd]:IMTI
Organizational Unit Name (eg, section) []:IMTI
Common Name (eg, your name or your server's hostname) []:ivan
Email Address []:hakerivan@163.com
```

步骤 06 使用 openssl 命令生成服务器端的密钥对。

```
[root@localhost newcerts]# openssl req -newkey rsa:2048 -days 1000 -nodes
-keyout server-key.pem > server-req.pem
```



```

Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'server-key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:zg
State or Province Name (full name) [Berkshire]:hb
Locality Name (eg, city) [Newbury]:xf
Organization Name (eg, company) [My Company Ltd]:imti
Organizational Unit Name (eg, section) []:imti
Common Name (eg, your name or your server's hostname) []:ivan
Email Address []:hakerivan@163.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
An optional company name []:imti
[root@localhost newcerts]#
[root@localhost newcerts]# openssl x509 -req -in server-req.pem -days 1000 -CA
ca-cert.pem -CAkey ca-key.pem -set_serial 01 > server-cert.pem
Signature ok
subject=/C=zg/ST=hb/L=xf/O=imti/OU=imti/CN=ivan/emailAddress=hakerivan@163.
com
Getting CA Private Key

```

步骤 07 使用 openssl 命令生成客户端的密钥对。

```

[root@localhost newcerts]# openssl x509 -req -in server-req.pem -days 1000 -CA
ca-cert.pem -CAkey ca-key.pem -set_serial 01 > server-cert.pem
Signature ok
subject=/C=zg/ST=hb/L=xf/O=imti/OU=imti/CN=ivan/emailAddress=hakerivan@163.
com
Getting CA Private Key
[root@localhost newcerts]#
[root@localhost newcerts]#
[root@localhost newcerts]# openssl req -newkey rsa:2048 -days 1000 -nodes
-keyout client-key.pem > client-req.pem
Generating a 2048 bit RSA private key
.....+++

```

```

.....+++
writing new private key to 'client-key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:zg
State or Province Name (full name) [Berkshire]:hb
Locality Name (eg, city) [Newbury]:xf
Organization Name (eg, company) [My Company Ltd]:imti
Organizational Unit Name (eg, section) []:imti
Common Name (eg, your name or your server's hostname) []:ivan
Email Address []:hakerivan@163.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:password
An optional company name []:imti

[root@localhost newcerts]# openssl x509 -req -in client-req.pem -days 1000 -CA
ca-cert.pem -CAkey ca-key.pem -set_serial 01 > client-cert.pem
Signature ok
subject=/C=zg/ST=hb/L=xf/O=imti/OU=imti/CN=ivan/emailAddress=hakerivan@163.
com
Getting CA Private Key
[root@localhost newcerts]#

```

步骤 08 使用 ls 命令查看下生成的 CA 密钥文件、服务器密钥文件和客户端密钥文件。

```

[root@localhost newcerts]# ls -al /etc/mysql/newcerts
total 40
drwxr-xr-x 2 root root 4096 2012-06-16 16:33 .
drwxr-xr-x 3 root root 4096 2012-06-16 16:17 ..
-rw-r--r-- 1 root root 1562 2012-06-16 16:22 ca-cert.pem
-rw-r--r-- 1 root root 1675 2012-06-16 16:18 ca-key.pem
-rw-r--r-- 1 root root 1237 2012-06-16 16:33 client-cert.pem
-rw-r--r-- 1 root root 1679 2012-06-16 16:32 client-key.pem
-rw-r--r-- 1 root root 1086 2012-06-16 16:32 client-req.pem
-rw-r--r-- 1 root root 1237 2012-06-16 16:30 server-cert.pem
-rw-r--r-- 1 root root 1679 2012-06-16 16:27 server-key.pem
-rw-r--r-- 1 root root 1086 2012-06-16 16:27 server-req.pem

```



```
[root@localhost newcerts]#
[root@localhost newcerts]# chmod 777 ./*.*
```

步骤 09 把生成的 CA 密钥文件和客户端的密钥文件传递给客户端，这里使用 windows 客户端。把 client-cert.pem、client-key.pem、client-req.pem、ca-cert.pem、ca-key.pem 传递给客户端。客户端需要配置一下 my.ini 文件，设置如下：

```
ssl-ca="C:\Program Files\MySQL\MySQL Server 5.6\SSL_key\ca-cert.pem"
ssl-cert="C:\Program Files\MySQL\MySQL Server 5.6\SSL_key\client-cert.pem"
ssl-key="C:\Program Files\MySQL\MySQL Server 5.6\SSL_key\client-key.pem"
```

步骤 10 服务器使用含 REQUIRE SSL 子句的 GRANT 语句在服务器上创建一个用户，然后使用该用户来连接服务器，服务器和客户端均应该支持 SSL，如下：

```
[root@localhost newcerts]#grant all on *.* to test2@'192.168.0.23' identified
by '123'require ssl;
Query OK, 0 rows affected (0.00 sec)
```

步骤 11 客户端测试 SSL 连接，结果如下：

```
C:/>mysql -utest2 -p -h192.168.0.5
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or /g.
Your MySQL connection id is 13
Server version: 5.1.30-log Source distribution

Type 'help;' or '/h' for help. Type '/c' to clear the buffer.

mysql> status;
-----
mysql Ver 14.12 Distrib 5.0.45, for Win32 (ia32)

Connection id:          13
Current database:
Current user:            root@wh88.wswtek.com
SSL:                    Cipher in use is DHE-RSA-AES256-SHA
Using delimiter:        ;
Server version:         5.1.30-log Source distribution
Protocol version:       10
Connection:             192.168.2.41 via TCP/IP
Server characterset:    latin1
Db characterset:        latin1
Client characterset:    utf8
Conn. characterset:     utf8
TCP port:               3309
Uptime:                 20 min 43 sec
```

```
Threads: 1 Questions: 27 Slow queries: 0 Opens: 22 Flush tables: 2 Open tables: 7
Queries per second avg: 0.21
-----
```

值得注意的是, MySQL 数据库如果设置 SSL 安全连接, 那么所有的用户默认都用 SSL 安全连接。如果不想使用 SSL 连接, 可以在连接命令后面加上 “-ssl=0” 即可, 例如:

```
[root@localhost newcerts]#mysql -utest2 -p -ssl=0
```

2.7 综合管理用户权限

本章详细介绍了 MySQL 如何管理用户对服务器的访问控制和 root 用户如何对每一个账户授予权限。这些被授予的权限分为不同的层级, 可以是全局层级、数据库层级、表层级或者是列层级等, 读者可以灵活地将混合权限授予各个需要的用户。通过本章的内容, 读者将学会如何创建账户, 如何对账户授权、如何收回权限以及如何删除账户。下面的综合实例将帮助读者建立执行这些操作的能力。

1. 案例目的

掌握创建用户和授权的方法。

2. 案例操作过程

步骤 01 打开 MySQL 客户端工具, 输入登录命令, 登录 MySQL。

```
C:\>mysql -u root -p
Enter password: **
```

输入正确密码, 按回车键, 出现欢迎信息表示登录成功。

步骤 02 选择 mysql 数据库为当前数据库。

```
MySQL> use mysql;
Database changed
```

出现 Database changed 信息表明切换数据库成功。

步骤 03 创建新账户, 用户名称为 newAdmin, 密码为 pw1, 允许其从本地主机访问 MySQL。

使用 GRANT 语句创建新账户, 创建过程如下:

```
MySQL> GRANT SELECT, UPDATE(id, name, age)
-> ON test_db.person_old
-> TO 'newAdmin'@'localhost' IDENTIFIED BY 'pw1'
-> WITH MAX_CONNECTIONS_PER_HOUR 30;
Query OK, 0 rows affected (0.06 sec)
```


提示消息可以看到，语句执行成功。

步骤 04 分别从 user 表中查看新账户的账户信息，从 tables_priv 和 columns_priv 表中查看权限信息。

用户账户创建完成之后，账户信息已经保存在 user 表中，权限信息则分别保存在 tables_priv 和 columns_priv 中，查询 user 名称为 newAdmin 的账户信息，执行过程如下：

```
SELECT host, user, select_priv, update_priv FROM user WHERE user='newAdmin';

SELECT host, db, user, table_name, table_priv, column_priv
FROM tables_priv WHERE user='newAdmin';

SELECT host, db, user, table_name, column_name, column_priv
FROM columns_priv WHERE user='newAdmin';
```

3 条 SQL 语句的查询结果分别如下：

```
MySQL> SELECT host, user, select_priv, update_priv FROM user WHERE
user='newAdmin';
+-----+-----+-----+-----+
| host      | user      | select_priv | update_priv |
+-----+-----+-----+-----+
| localhost | newAdmin  | N           | N           |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

MySQL> SELECT host, db, user, table_name, table_priv, column_priv
-> FROM tables_priv WHERE user='newAdmin';
+-----+-----+-----+-----+-----+-----+
| host      | db  | user      | table_name | table_priv | column_priv |
+-----+-----+-----+-----+-----+-----+
| localhost | test | newAdmin  | person     | Select     | Update      |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

MySQL> SELECT host, db, user, table_name, column_name, column_priv
-> FROM columns_priv WHERE user='newAdmin';
+-----+-----+-----+-----+-----+-----+
| host      | db  | user      | table_name | column_name | column_priv |
+-----+-----+-----+-----+-----+-----+
| localhost | test | newAdmin  | person     | id          | Update      |
| localhost | test | newAdmin  | person     | name        | Update      |
| localhost | test | newAdmin  | person     | age         | Update      |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

步骤 05 使用 SHOW GRANTS 语句查看 newAdmin 的权限信息。

查看 newAdmin 账户的权限信息，输入语句如下：

```
SHOW GRANTS FOR 'newAdmin'@'localhost';
```

执行结果如下：

```
+-----+
| Grants for newAdmin@localhost |
+-----+
| GRANT USAGE ON *.* TO 'newAdmin'@'localhost' IDENTIFIED BY PASSWORD '*2B602296A79E0A8784ACC5C88D92E46588CCA3C3' WITH MAX_CONNECTIONS_PER_HOUR 30 |
| GRANT SELECT, UPDATE (age, id, name) ON `test`.`person` TO 'newAdmin'@'localhost' |
+-----+
2 rows in set (0.00 sec)
```

步骤 06 使用 newAdmin 用户登录 MySQL。

退出当前登录，使用 EXIT 命令，语句如下：

```
MySQL> exit
Bye
```

使用 newAdmin 账户登录 MySQL，语句如下：

```
C:\>MySQL -u newAdmin -p
Enter password: ***
```

输入密码正确后，出现“mysql>”提示符，登录成功。

步骤 07 使用 newAdmin 用户查看 test_db 数据库中 person_dd 表中的数据。

newAdmin 用户被授予 test 数据库中 person 表中 3 个字段上的查询权限，因此可以执行 SELECT 语句查看这几个字段的值，执行过程如下：

```
MySQL> SELECT * FROM test_db.person_dd LIMIT 5;
+----+-----+-----+-----+
| id | name  | age  | info      |
+----+-----+-----+-----+
| 1  | Green | 21   | Lawyer    |
| 2  | Suse  | 22   | dancer    |
| 3  | Mary  | 24   | Musician  |
| 4  | Willam | 20  | sports man |
| 5  | Laura | 25   | NULL      |
+----+-----+-----+-----+
5 rows in set (0.00 sec)
```

可以看到，查询结果显示了表中的前 5 条记录。

步骤08 使用 newAdmin 用户向 person_dd 表中插入一条新记录，查看语句执行结果。

插入新记录，输入语句如下：

```
INSERT INTO test_db.person_dd(name, age, info) VALUES('gaga', 30);
```

执行结果如下：

```
ERROR 1142 (42000): INSERT command denied to user 'newAdmin'@'localhost' for table 'person'
```

可以看到，语句不能执行，错误信息表明 newAdmin 用户不能对 person 表进行插入操作。因此，用户不可以执行没有被授权的操作语句。

步骤09 退出当前登录，使用 root 用户重新登录，收回 newAdmin 账户的权限。

输入退出命令：

```
exit
```

重新以 root 用户登录 MySQL，并选择 mysql 数据库为当前数据库。

输入语句收回 newAdmin 账户的权限，执行过程如下：

```
REVOKE SELECT, UPDATE ON test.person FROM 'newAdmin'@'localhost';
```

执行结果如下：

```
MySQL> REVOKE SELECT, UPDATE ON test.person FROM 'newAdmin'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

步骤10 删除 newAdmin 的账户信息。

删除指定账户，可以使用 DROP USER 语句，输入如下：

```
DROP USER 'newAdmin'@'localhost';
```

语句执行成功之后，tables_priv 和 columns_priv 中相关的记录将被删除。

2.8 小结

数据库的权限系统和数据库安全对企业来讲是非常重要的，本章重点讲解了 MySQL 系统中权限表、账户管理和权限管理中可能存在的一些安全隐患。通过例子讲解了数据库系统和操作系统对数据库安全隐患方面需要注意的地方，最后讲解了 SSL 安全连接 MySQL 数据库，防止数据在网络传输中被窃取。本章还讲解了数据库如何授权的知识，读者可以根据需求选读其中的章节。

第 3 章

◀ 数据备份与还原 ▶

尽管采取了一些管理措施来保证数据库的安全,但是不确定的意外情况总是有可能造成数据的损失,例如意外的停电、管理员不小心的操作失误都可能会造成数据的丢失。保证数据安全最重要的一个措施是确保对数据进行定期备份。如果数据库中的数据丢失或者出现错误,可以使用备份的数据进行还原,这样就尽可能地降低了意外原因导致的损失。MySQL 提供了多种方法对数据进行备份和还原。本章将介绍数据备份、数据还原、数据迁移和数据导入导出的相关知识。

3.1 数据备份

数据备份是数据库管理员非常重要的工作。系统意外崩溃或者硬件的损坏都可能导致数据库的丢失,因此 MySQL 管理员应该定期地备份数据库,使得在意外情况发生时,尽可能减少损失。本节将介绍数据备份的 3 种方法。

3.1.1 使用 mysqldump 命令备份

mysqldump 是 MySQL 提供的一个非常有用的数据库备份工具。mysqldump 命令执行时,可以将数据库备份成一个文本文件,该文件中实际上包含了多个 CREATE 和 INSERT 语句,使用这些语句可以重新创建表和插入数据。

mysqldump 备份数据库语句的基本语法格式如下:

```
mysqldump -u user -h host -p password dbname [tblname, [tblname...]] > filename.sql
```

user 表示用户名称; host 表示登录用户的主机名称; password 为登录密码; dbname 为需要备份的数据库名称; tblname 为 dbname 数据库中需要备份的数据表,可以指定多个需要备份的表。右箭头符号“>”告诉 mysqldump 将备份数据表的定义和数据写入备份文件; filename.sql 为备份文件的名称。

1. 使用 mysqldump 备份单个数据库中的所有表

【例 3.1】使用 mysqldump 命令备份数据库中的所有表。

为了更好地理解 `mysqldump` 工具如何工作，本章给出一个完整的数据库例子。首先登录 MySQL，按下面数据库结构创建 `booksDB` 数据库和各个表，并插入数据记录。数据库和表定义如下：

```
CREATE DATABASE booksDB;
use booksDB;

CREATE TABLE books
(
bk_id INT NOT NULL PRIMARY KEY,
bk_title VARCHAR(50) NOT NULL,
copyright YEAR NOT NULL
);
INSERT INTO books
VALUES (11078, 'Learning MySQL', 2010),
(11033, 'Study Html', 2011),
(11035, 'How to use php', 2003),
(11072, 'Teach yourself javascript', 2005),
(11028, 'Learning C++', 2005),
(11069, 'MySQL professional', 2009),
(11026, 'Guide to MySQL 5.6', 2008),
(11041, 'Inside VC++', 2011);

CREATE TABLE authors
(
auth_id INT NOT NULL PRIMARY KEY,
auth_name VARCHAR(20),
auth_gender CHAR(1)
);
INSERT INTO authors
VALUES (1001, 'WriterX' , 'f'),
(1002, 'WriterA' , 'f'),
(1003, 'WriterB' , 'm'),
(1004, 'WriterC' , 'f'),
(1011, 'WriterD' , 'f'),
(1012, 'WriterE' , 'm'),
(1013, 'WriterF' , 'm'),
(1014, 'WriterG' , 'f'),
(1015, 'WriterH' , 'f');

CREATE TABLE authorbook
(
auth_id INT NOT NULL,
bk_id INT NOT NULL,
PRIMARY KEY (auth_id, bk_id),
```

```
FOREIGN KEY (auth_id) REFERENCES authors (auth_id),
FOREIGN KEY (bk_id) REFERENCES books (bk_id)
);
```

```
INSERT INTO authorbook
VALUES (1001, 11033), (1002, 11035), (1003, 11072), (1004, 11028),
(1011, 11078), (1012, 11026), (1012, 11041), (1014, 11069);
```

完成数据插入后打开操作系统命令行输入窗口，输入备份命令如下：

```
C:\>mysqldump -u root -p booksdb > C:/backup/booksdb_20130301.sql
Enter password: **
```

输入密码之后，MySQL 便对数据库进行了备份，在 C:\backup 文件夹下面查看刚才备份过的文件，使用文本查看器打开文件可以看到其部分文件内容大致如下：

```
-- MySQL dump 10.13 Distrib 5.6.10, for Win32 (x86)
--
-- Host: localhost    Database: booksDB
-- -----
-- Server version    5.6.10

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_
CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE=
'NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `authorbook`
--
DROP TABLE IF EXISTS `authorbook`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `authorbook` (
  `auth_id` int(11) NOT NULL,
  `bk_id` int(11) NOT NULL,
  PRIMARY KEY (`auth_id`,`bk_id`),
  KEY `bk_id` (`bk_id`),
```



```

CONSTRAINT `authorbook_ibfk_1` FOREIGN KEY (`auth_id`)
REFERENCES `authors` (`auth_id`),
CONSTRAINT `authorbook_ibfk_2` FOREIGN KEY (`bk_id`)
REFERENCES `books` (`bk_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `authorbook`
--

LOCK TABLES `authorbook` WRITE;
/*!40000 ALTER TABLE `authorbook` DISABLE KEYS */;
INSERT INTO `authorbook` VALUES
(1012,11026),(1004,11028),(1001,11033),(1002,11035),(1012,
11041),(1014,11069),(1003,11072),(1011,11078);
/*!40000 ALTER TABLE `authorbook` ENABLE KEYS */;
UNLOCK TABLES;
...
...省略部分内容
...
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
-- Dump completed on 2011-08-18 10:44:08

```

可以看到，备份文件包含了一些信息，文件开头首先表明了备份文件使用的 `mysqldump` 工具的版本号；然后是备份账户的名称和主机信息，以及备份的数据库的名称，最后是 MySQL 服务器的版本号，在这里为 5.6.10。

备份文件接下来的部分是一些 `SET` 语句，这些语句将一些系统变量值赋给用户定义变量，以确保被恢复的数据库的系统变量和原来备份时的变量相同，例如：

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
```

该 `SET` 语句将当前系统变量 `CHARACTER_SET_CLIENT` 的值赋给用户定义变量 `@OLD_CHARACTER_SET_CLIENT`。其他变量与此类似。

备份文件的最后几行 MySQL 使用 `SET` 语句恢复服务器系统变量原来的值，例如：

```
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

该语句将用户定义的变量@OLD_CHARACTER_SET_CLIENT 中保存的值赋给实际的系统变量 CHARACTER_SET_CLIENT。

备份文件中以“--”字符开头的语句为行为注释语句；以“/*!”开头、“*/”结尾的语句为可执行的 MySQL 注释语句，这些语句可以被 MySQL 执行，但在其他数据库管理系统将被作为注释语句忽略，这可以提高数据库的可移植性。

另外注意到，备份文件开始的一些语句以数字开头，这些数字代表了 MySQL 版本号，该数字告诉我们，这些语句只有在指定的 MySQL 版本或者比该版本高的情况下才能执行。例如 40101，表明这些语句只有在 MySQL 版本号为 4.01.01 或者更高的条件下才可以被执行。

2. 使用 mysqldump 备份数据库中的某个表

在前面 mysqldump 语法中介绍过，mysqldump 还可以备份数据中的某个表，其语法格式为：

```
mysqldump -u user -h host -p dbname [tblname, [tblname...]] > filename.sql
```

tblname 表示数据库中的表名，多个表名之间用空格隔开。

备份表和备份数据库中所有表的语句中不同的地方在于，要在数据库名称 dbname 之后指定需要备份的表名称。

【例 3.2】备份 booksDB 数据库中的 books 表，输入语句如下：

```
mysqldump -u root -p booksDB books > C:/backup/books_20130301.sql
```

该语句创建名称为 books_20130301.sql 的备份文件，文件中包含了前面介绍的 SET 语句等内容，不同的是，该文件只包含 books 表的 CREATE 和 INSERT 语句。

3. 使用 mysqldump 备份多个数据库

如果要使用 mysqldump 备份多个数据库，需要使用--databases 参数。备份多个数据库的语句格式如下：

```
mysqldump -u user -h host -p --databases [dbname, [dbname...]] > filename.sql
```

使用--databases 参数之后，必须指定至少一个数据库的名称，多个数据库名称之间用空格隔开。

【例 3.3】使用 mysqldump 备份 booksDB 和 test 数据库，输入语句如下：

```
mysqldump -u root -p --databases booksDB test>
C:\backup\books_testDB_20130301.sql
```

该语句创建名称为 books_testDB_20130301.sql 的备份文件，文件中包含了创建两个数据库 booksDB 和 test_db 所必须的所有语句。

另外，使用--all-databases 参数可以备份系统中所有的数据库，语句如下：


```
mysqldump -u user -h host -p --all-databases > filename.sql
```

使用--all-databases 参数时，不需要指定数据库名称。

【例 3.4】使用 mysqldump 备份服务器中的所有数据库，输入语句如下：

```
mysqldump -u root -p --all-databases > C:/backup/alldbinMySQL.sql
```

该语句创建名称为 alldbinMySQL.sql 的备份文件，文件中包含了对系统中所有数据库的备份信息。

提示

如果在服务器上进行备份，并且表均为 MyISAM 表，应考虑使用 mysqlhotcopy，因为可以更快地进行备份和恢复。

mysqldump 还有一些其他选项可以用来制定备份过程，例如--opt 选项，该选项将打开--quick、--add-locks、--extended-insert 等多个选项。使用--opt 选项可以提供最快速的数据库转储。

mysqldump 其他常用选项如下：

- --add-drop-database，在每个 CREATE DATABASE 语句前添加 DROP DATABASE 语句。
- --add-drop-tables，在每个 CREATE TABLE 语句前添加 DROP TABLE 语句。
- --add-locking，用 LOCK TABLES 和 UNLOCK TABLES 语句引用每个表转储。重载转储文件时插入得更快。
- --all--database，-A 转储所有数据库中的所有表。与使用--database 选项相同，在命令行中命名所有数据库。
- --comments[=0|1]，如果设置为 0，禁止转储文件中的其他信息，例如程序版本、服务器版本和主机。--skip-comments 与--comments=0 的结果相同。默认值为 1，即包括额外信息。
- --compact，产生少量输出。该选项禁用注释并启用--skip-add-drop-tables、--no-set-names、--skip-disable-keys 和--skip-add-locking 选项。
- --compatible=name，产生与其他数据库系统或旧的 MySQL 服务器更兼容的输出。值可以为 ansi、mysql323、mysql40、postgresql、oracle、mssql、db2、maxdb、no_key_options、no_tables_options 或者 no_field_options。
- --complete-insert，-c 使用包括列名的完整的 INSERT 语句。
- ---debug[=debug_options]，-# [debug_options] 写调试日志。
- --delete，-D 导入文本文件前清空表。
- --default-character-set=charset，使用 charsets 默认字符集。如果没有指定，mysqldump 使用 utf8。
- --delete-master-logs，在主复制服务器上，完成转储操作后删除二进制日志。该选项自动启用--master-data。

- `--extended-insert`, `-e` 使用包括几个 VALUES 列表的多行 INSERT 语法。这样使转储文件更小, 重载文件时可以加速插入。
- `--flush-logs`, `-F` 开始转储前刷新 MySQL 服务器日志文件。该选项要求 RELOAD 权限。
- `--force`, `-f` 在表转储过程中, 即使出现 SQL 错误也继续。
- `--lock-all-tables`, `-x` 对所有数据库中的所有表加锁。在整体转储过程中通过全局锁定来实现。该选项自动关闭 `--single-transaction` 和 `--lock-tables`。
- `--lock-tables`, `-l` 开始转储前锁定所有表。用 READ LOCAL 锁定表以允许并行插入 MyISAM 表。对于事务表 (例如 InnoDB 和 BDB), `--single-transaction` 是一个更好的选项, 因为它根本不需要锁定表。
- `--no-create-db`, `-n` 该选项禁用 CREATE DATABASE /*!32312 IF NOT EXISTS*/ db_name 语句, 如果给出 `--database` 或 `--all--database` 选项, 则包含到输出中。
- `--no-create-info`, `-t` 只导出数据, 而不添加 CREATE TABLE 语句。
- `--no-data`, `-d` 不写表的任何行信息, 只转储表的结构。
- `--opt`, 该选项是速记, 等同于指定 `--add-drop-tables--add-locking`, `--create-option`, `--disable-keys--extended-insert`, `--lock-tables-quick` 和 `--set-charset`。它可以快速进行转储操作并产生一个能很快装入 MySQL 服务器的转储文件。该选项默认开启, 但可以用 `--skip-opt` 禁用。要想禁用使用 `-opt` 启用的选项, 可以使用 `--skip` 形式, 例如 `--skip-add-drop-tables` 或 `--skip-quick`。
- `--password[=password]`, `-p[password]`, 当连接服务器时使用的密码。如果使用短选项形式 (`-p`), 选项和密码之间不能有空格。如果在命令行中 `--password` 或 `-p` 选项后面没有密码值, 则提示输入一个密码。
- `--port=port_num`, `-P port_num`, 用于连接的 TCP/IP 端口号。
- `--protocol={TCP | SOCKET | PIPE | MEMORY}`, 使用的连接协议。
- `--replace`, `-r` `--replace` 和 `--ignore`, 这些选项控制替换或复制唯一键值已有记录的输入记录的处理。如果指定 `--replace`, 新行替换有相同的唯一键值的已有行; 如果指定 `--ignore`, 复制已有的唯一键值的输入行被跳过。如果不指定这两个选项, 当发现一个复制键值时会出现一个错误, 并且忽视文本文件的剩余部分。
- `--silent`, `-s` 沉默模式。只有出现错误时才输出。
- `--socket=path`, `-S path` 当连接 localhost 时使用的套接字文件 (为默认主机)。
- `--user=user_name`, `-u user_name` 当连接服务器时 MySQL 使用的用户名。
- `--verbose`, `-v` 冗长模式。打印出程序操作的详细信息。
- `--version`, `-V` 显示版本信息并退出。
- `--xml`, `-X` 产生 XML 输出。

mysqldump 提供许多选项, 包括用于调试和压缩的, 在这里只是列举最有用的。运行帮助命令 `mysqldump --help`, 可以获得特定版本的完整选项列表。

提示

如果运行 `mysqldump` 没有 `--quick` 或 `--opt` 选项, `mysqldump` 在转储结果前将整个结果集装入内存。如果转储大数据库, 可能会出现内存问题。该选项默认启用, 但可以用 `--skip-opt` 禁用。如果使用最新版本的 `mysqldump` 程序备份数据, 并用于还原到比较旧版本的 MySQL 服务器中, 则不要使用 `--opt` 或 `-e` 选项。

3.1.2 直接复制整个数据库目录

因为 MySQL 表保存为文件方式, 所以可以直接复制 MySQL 数据库的存储目录及文件进行备份。MySQL 的数据库目录位置不一定相同, 在 Windows 平台下, MySQL 5.6 存放数据库的目录通常默认为 “C:\Documents and Settings\All Users\Application Data\MySQL\MySQL Server 5.6\data” 或者其他用户自定义目录; 在 Linux 平台下, 数据库目录位置通常为 `/var/lib/mysql/`, 不同 Linux 版本下目录会有所不同, 读者应在自己使用的平台下查找该目录。

这是一种简单、快速、有效的备份方式。要想保持备份的一致性, 备份前需要对相关表执行 `LOCK TABLES` 操作, 然后对表执行 `FLUSH TABLES`。这样当复制数据库目录中的文件时, 允许其他客户继续查询表。需要 `FLUSH TABLES` 语句来确保开始备份前将所有激活的索引页写入硬盘。当然, 也可以停止 MySQL 服务再进行备份操作。

这种方法虽然简单, 但并不是最好的方法。因为这种方法对 InnoDB 存储引擎的表不适用。使用这种方法备份的数据最好还原到相同版本的服务器中, 不同的版本可能不兼容。

提示

在 MySQL 版本号中, 第一个数字表示主版本号, 主版本号相同的 MySQL 数据库文件格式相同。

3.1.3 使用 `mysqlhotcopy` 工具快速备份

`mysqlhotcopy` 是一个 Perl 脚本, 最初由 Tim Bunce 编写并提供。它使用 `LOCK TABLES`、`FLUSH TABLES` 和 `cp` 或 `scp` 来快速备份数据库。它是备份数据库或单个表的最快的途径, 但它只能运行在数据库目录所在的机器上, 并且只能备份 `MyISAM` 和 `ARCHIVE` 类型的表。`mysqlhotcopy` 在 Unix 系统中运行。

`mysqlhotcopy` 命令语法格式如下:

```
mysqlhotcopy db_name_1, ..., db_name_n /path/to/new_directory
```

`db_name_1,...,db_name_n` 分别为需要备份的数据库的名称; `/path/to/new_directory` 指定备份文件目录。

【例 3.5】 使用 `mysqlhotcopy` 备份 `test` 数据库到 `/usr/backup` 目录下, 输入语句如下:

```
mysqlhotcopy -u root -p test /usr/backup
```

要想执行 `mysqlhotcopy`, 必须可以访问备份的表文件, 具有那些表的 `SELECT` 权限、`RELOAD` 权限 (以便能够执行 `FLUSH TABLES`) 和 `LOCK TABLES` 权限。

提示

mysqlhotcopy 只是将表所在的目录复制到另一个位置，只能用于备份 MyISAM 和 ARCHIVE 表。备份 InnoDB 类型的数据表时会出现错误信息。由于它复制本地格式的文件，故也不能移植到其他硬件或操作系统下。

3.2 数据还原

管理人员操作的失误、计算机故障以及其他意外情况，都会导致数据的丢失和破坏。当数据丢失或意外破坏时，可以通过还原已经备份的数据尽量减少数据丢失和破坏造成的损失。本节将介绍数据还原的方法。

3.2.1 使用 MySQL 命令还原

对于已经备份的包含 CREATE、INSERT 语句的文本文件，可以使用 mysql 命令导入到数据库中。本小节将介绍 MySQL 命令导入 SQL 文件的方法。

备份的 SQL 文件中包含 CREATE、INSERT 语句（有时也会有 DROP 语句）。MySQL 命令可以直接执行文件中的这些语句。其语法如下：

```
mysql -u user -p [dbname] < filename.sql
```

user 是执行 backup.sql 中语句的用户名；-p 表示输入用户密码；dbname 是数据库名。如果 filename.sql 文件为 mysqldump 工具创建的包含创建数据库语句的文件，执行的时候不需要指定数据库名。

【例 3.6】使用 MySQL 命令将 C:\backup\booksdb_20130301.sql 文件中的备份导入到数据库中，输入语句如下：

```
mysql -u root -p booksDB < C:/backup/booksdb_20130301.sql
```

执行该语句前，必须先在 MySQL 服务器中创建 booksDB 数据库，如果不存在恢复过程将会出错。命令执行成功之后 booksdb_20130301.sql 文件中的语句就会在指定的数据库中恢复以前的表。

如果已经登录 MySQL 服务器，还可以使用 source 命令导入 SQL 文件。source 语句语法如下：

```
source filename
```

【例 3.7】使用 root 用户登录到服务器，然后使用 source 导入本地的备份文件 booksdb_20110101.sql，输入语句如下：

```
--选择要恢复到的数据库
mysql> use booksDB;
```



```
Database changed
```

```
--使用 source 命令导入备份文件
```

```
mysql> source C:\backup\booksDB_20130301.sql
```

命令执行后，会列出备份文件 booksDB_20130301.sql 中每一条语句的执行结果。source 命令执行成功后，booksDB_20130301.sql 中的语句会全部导入到现有数据库中。

提示

执行 source 命令前，必须使用 use 语句选择数据库。不然，恢复过程中会出现“ERROR 1046 (3D000): No database selected”的错误。

3.2.2 直接复制到数据库目录

如果数据库通过复制数据库文件备份，可以直接复制备份的文件到 MySQL 数据目录下实现还原。通过这种方式还原时，必须保存备份数据的数据库和待还原的数据库服务器的主版本号相同。而且这种方式只对 MyISAM 引擎的表有效，对于 InnoDB 引擎的表不可用。

执行还原以前关闭 mysql 服务，将备份的文件或目录覆盖 MySQL 的 data 目录，启动 mysql 服务。对于 Linux/Unix 操作系统来说，复制完文件需要将文件的用户和组更改为 mysql 运行的用户和组，通常用户是 mysql，组也是 mysql。

3.2.3 mysqlhotcopy 快速恢复

mysqlhotcopy 备份后的文件也可以用来恢复数据库，在 MySQL 服务器停止运行时，将备份的数据库文件复制到 MySQL 存放数据的位置（MySQL 的 data 文件夹），重新启动 MySQL 服务即可。如果以根用户执行该操作，必须指定数据库文件的所有者，输入语句如下：

```
chown -R mysql:mysql /var/lib/mysql/dbname
```

【例 3.8】从 mysqlhotcopy 复制的备份恢复数据库，输入语句如下：

```
cp -R /usr/backup/test usr/local/mysql/data
```

执行完该语句，重启服务器，MySQL 将恢复到备份状态。

提示

如果需要恢复的数据库已经存在，则在使用 DROP 语句删除已经存在的数据库之后，恢复才能成功。另外 MySQL 不同版本之间必须兼容，恢复之后的数据才可以使用。

3.3 数据库迁移

数据库迁移就是把数据从一个系统移动到另一个系统上。数据迁移有以下原因：

- 需要安装新的数据库服务器。
- MySQL 版本更新。
- 数据库管理系统的变更（如从 Microsoft SQL Server 迁移到 MySQL）。

本小节将讲解数据库迁移的方法。

3.3.1 相同版本的 MySQL 数据库之间的迁移

相同版本的 MySQL 数据库之间的迁移就是在主版本号相同的 MySQL 数据库之间进行数据库移动。迁移过程其实就是在源数据库备份和目标数据库还原过程的组合。

在讲解数据库备份和还原时，已经知道最简单的方式是通过复制数据库文件目录，但是此种方法只适用于 MyISAM 引擎的表。而对于 InnoDB 表，不能用直接复制文件的方式备份数据库，因此最常用和最安全的方式是使用 `mysqldump` 命令导出数据，然后在目标数据库服务器使用 `mysql` 命令导入。

【例 3.9】将 `www.abc.com` 主机上的 MySQL 数据库全部迁移到 `www.bcd.com` 主机上。在 `www.abc.com` 主机上执行的命令如下：

```
mysqldump -h www.abc.com -u root -ppassword dbname |  
mysql -h www.bcd.com -uroot -ppassword
```

`mysqldump` 导入的数据直接通过管道符“|”，传给 `mysql` 命令导入到主机 `www.bcd.com` 数据库中，`dbname` 为需要迁移的数据库名称，如果要迁移全部的数据库，可使用参数 `--all-databases`。

3.3.2 不同版本的 MySQL 数据库之间的迁移

因为数据库升级等原因，需要将较旧版本 MySQL 数据库中的数据迁移到较新版本的数据库中。MySQL 服务器升级时，需要先停止服务，然后卸载旧版本，并安装新版的 MySQL，这种更新方法很简单，如果想保留旧版本中的用户访问控制信息，则需要备份 MySQL 中的 `mysql` 数据库，在新版本 MySQL 安装完成之后，重新读入 `mysql` 备份文件中的信息。

旧版本与新版本的 MySQL 可能使用不同的默认字符集，例如 MySQL 4.x 中大多使用 `latin1` 作为默认字符集，而 MySQL 5.x 的默认字符集为 `utf8`。如果数据库中有中文数据的，迁移过程中需要对默认字符集进行修改，不然可能无法正常显示结果。

新版本会对旧版本有一定兼容性。从旧版本的 MySQL 向新版本的 MySQL 迁移时，对于 MyISAM 引擎的表，可以直接复制数据库文件，也可以使用 `mysqlhotcopy` 工具、`mysqldump` 工具。对于 InnoDB 引擎的表，一般只能使用 `mysqldump` 将数据导出。然后使用 `mysql` 命令导入到目标服务器上。从新版本向旧版本 MySQL 迁移数据时要特别小心，最好使用 `mysqldump` 命令导出，然后导入目标数据库中。

3.3.3 不同数据库之间的迁移

不同类型的数据库之间的迁移，是指把 MySQL 的数据库转移到其他类型的数据库，例如从 MySQL 迁移到 Oracle，从 Oracle 迁移到 MySQL，从 MySQL 迁移到 SQL Server 等。

迁移之前，需要了解不同数据库的架构，比较它们之间的差异。不同数据库中定义相同类型的数据的关键字可能会不同。例如，MySQL 中日期字段分为 DATE 和 TIME 两种，而 Oracle 日期字段只有 DATE。另外，数据库厂商并没有完全按照 SQL 标准来设计数据库系统，导致不同的数据库系统的 SQL 语句有差别。例如，MySQL 几乎完全支持标准 SQL 语言，而 Microsoft SQL Server 使用的是 T-SQL 语言，T-SQL 中有一些非标准的 SQL 语句，因此在迁移时必须对这些语句进行语句映射处理。

数据库迁移可以使用一些工具，例如在 Windows 系统下，可以使用 MyODBC 实现 MySQL 和 SQL Server 之间的迁移。MySQL 官方提供的工具 MySQL Migration Toolkit 也可以在不同数据库间进行数据迁移。

3.4 表的导出和导入

有时会需要将 MySQL 数据库中的数据导出到外部存储文件中，MySQL 数据库中的数据可以导出成 SQL 文本文件、xml 文件或者 html 文件。同样这些导出文件也可以导入到 MySQL 数据库中。本小节将介绍数据导出和导入的常用方法。

3.4.1 使用 SELECT...INTO OUTFILE 导出文本文件

MySQL 数据库导出数据时，允许使用包含导出定义的 SELECT 语句进行数据的导出操作。该文件被创建到服务器主机上，因此必须拥有文件写入权限（FILE 权限），才能使用此语法。“SELECT...INTO OUTFILE 'filename'”形式的 SELECT 语句可以把被选择的行写入一个文件中，filename 不能是一个已经存在的文件。SELECT...INTO OUTFILE 语句基本格式如下：

```
SELECT columnlist FROM table WHERE condition INTO OUTFILE 'filename'
[OPTIONS]
```

--OPTIONS 选项

```
    FIELDS TERMINATED BY 'value'
FIELDS [OPTIONALLY] ENCLOSED BY 'value'
FIELDS ESCAPED BY 'value'
    LINES STARTING BY 'value'
    LINES TERMINATED BY 'value'
```

可以看到 SELECT columnlist FROM table WHERE condition 为一个查询语句，查询结果返

回满足指定条件的一条或多条记录；INTO OUTFILE 语句的作用就是把前面 SELECT 语句查询出来的结果导出到名称为“filename”的外部文件中。[OPTIONS]为可选参数选项，OPTIONS 部分的语法包括 FIELDS 和 LINES 子句，其可能的取值有：

- FIELDS TERMINATED BY 'value': 设置字段之间的分隔字符，可以为单个或多个字符，默认情况下为制表符“\t”。
- FIELDS [OPTIONALLY] ENCLOSED BY 'value': 设置字段的包围字符，只能为单个字符，如果使用了 OPTIONALLY，则只有 CHAR 和 VARCHAR 等字符数据字段被包括。
- FIELDS ESCAPED BY 'value': 设置如何写入或读取特殊字符，只能为单个字符，即设置转义字符，默认值为“\”。
- LINES STARTING BY 'value': 设置每行数据开头的字符，可以为单个或多个字符，默认情况下不使用任何字符。
- LINES TERMINATED BY 'value': 设置每行数据结尾的字符，可以为单个或多个字符，默认值为“\n”。

FIELDS 和 LINES 两个子句都是自选的，但是如果两个都被指定了，FIELDS 必须位于 LINES 的前面。

SELECT...INTO OUTFILE 语句可以非常快速地把一个表转储到服务器上。如果想要在服务器主机之外的部分客户主机上创建结果文件，不能使用 SELECT...INTO OUTFILE。在这种情况下，应该在客户主机上使用像“mysql -e "SELECT ..." > file_name”这样的命令，来生成文件。

SELECT...INTO OUTFILE 是 LOAD DATA INFILE 的补语。用于语句的 OPTIONS 部分的语法包括部分 FIELDS 和 LINES 子句，这些子句与 LOAD DATA INFILE 语句同时使用。

【例 3.10】使用 SELECT...INTO OUTFILE 将 test 数据库中的 person 表中的记录导出到文本文件，输入命令如下：

```
SELECT * FROM test.person INTO OUTFILE "C:\person0.txt";
```

由于指定了 INTO OUTFILE 子句，SELECT 将查询出来的 3 个字段的值保存到 C:\person0.txt 文件中，打开文件内容如下：

```
1  Green  21  Lawyer
2  Suse   22  dancer
3  Mary   24  Musician
4  Willam 20  sports man
5  Laura  25  \N
6  Evans  27  secretary
7  Dale   22  cook
8  Edison 28  singer
9  Harry  21  magician
10 Harriet 19  pianist
```

可以看到默认情况下，MySQL 使用制表符“\t”分隔不同的字段，字段没有被其他字符括起来。另外，在 Windows 平台下，使用记事本打开该文件，显示的格式与这里并不相同，这是因为 Windows 系统下回车换行为“\r\n”，默认换行符为“\n”，因此在 person.txt 中可能看到类似黑色方块的字符，所有的记录也会在同一行显示。

另外，注意到第 5 行中有一个字段值为“\N”，这表示该字段的值为 NULL。默认情况下，如果遇到 NULL 值，将会返回“\N”代表空值，反斜线“\”表示转义字符，如果使用 ESCAPED BY 选项，则 N 前面为指定的转义字符。

【例 3.11】使用 SELECT...INTO OUTFILE 将 test 数据库中的 person 表中的记录导出到文本文件，使用 FIELDS 选项和 LINES 选项，要求字段之间使用逗号“,”间隔，所有字段值用双引号括起来，定义转义字符为单引号“'”，执行的命令如下：

```
SELECT * FROM test.person INTO OUTFILE "C:\person1.txt"
FIELDS
TERMINATED BY ','
ENCLOSED BY '"'
ESCAPED BY '\''
LINES
TERMINATED BY '\r\n';
```

该语句将把 person 表中所有记录导入到 C 盘目录下的 person1.txt 文本文件中。

FIELDS TERMINATED BY ',' 表示字段之间用逗号分隔；ENCLOSED BY '"' 表示每个字段用双引号括起来；ESCAPED BY '\'' 表示将系统默认的转义字符替换为单引号；LINES TERMINATED BY '\r\n' 表示每行以回车换行符结尾，保证每一条记录占一行。

执行成功后，在目录 C:\ 下生成一个 person1.txt 文件，打开文件内容如下：

```
"1","Green","21","Lawyer"
"2","Suse","22","dancer"
"3","Mary","24","Musician"
"4","Willam","20","sports man"
"5","Laura","25",'N'
"6","Evans","27","secretary"
"7","Dale","22","cook"
"8","Edison","28","singer"
"9","Harry","21","magician"
"10","Harriet","19","pianist"
```

可以看到，所有的字段值都被双引号包括；第 5 条记录中空值的表示形式为“N”，即使使用单引号替换了反斜线转义字符。

【例 3.12】使用 SELECT...INTO OUTFILE 将 test 数据库中的 person 表中的记录导出到文本文件，使用 LINES 选项，要求每行记录以字符串“>”开始，以“<end>”字符串结尾，执行的命令如下：

```
SELECT * FROM test.person INTO OUTFILE "C:\person2.txt"
LINES
STARTING BY '> '
TERMINATED BY '<end>';
```

执行成功后，在目录 C:\ 下生成一个 person2.txt 文件，打开文件内容如下：

```
> 1 Green 21 Lawyer <end>> 2 Suse 22 dancer <end>> 3 Mary 24
Musician <end>> 4 Willam 20 sports man <end>> 5 Laura 25 \N <end>> 6
Evans 27 secretary <end>> 7 Dale 22 cook <end>> 8 Edison 28
singer <end>> 9 Harry 21 magician <end>> 10 Harriet 19 pianist <end>
```

可以看到，虽然将所有的字段值导出到文本文件中，但是所有的记录没有分行区分，出现这种情况是因为 TERMINATED BY 选项替换了系统默认的“\n”换行符，如果希望换行显示，则需要修改导出语句，输入下面语句：

```
SELECT * FROM test.person INTO OUTFILE "C:\person2.txt"
LINES
STARTING BY '> '
TERMINATED BY '<end>\r\n';
```

执行完语句之后，换行显示每条记录，结果如下：

```
> 1 Green 21 Lawyer <end>
> 2 Suse 22 dancer <end>
> 3 Mary 24 Musician <end>
> 4 Willam 20 sports man <end>
> 5 Laura 25 \N <end>
> 6 Evans 27 secretary <end>
> 7 Dale 22 cook <end>
> 8 Edison 28 singer <end>
> 9 Harry 21 magician <end>
> 10 Harriet 19 pianist <end>
```

3.4.2 用 mysqldump 命令导出文本文件

除了使用 SELECT... INTO OUTFILE 语句导出文本文件之外，还可以使用 mysqldump。本章开始介绍了使用 mysqldump 备份数据库，该工具不仅可以将数据导出为包含 CREATE、INSERT 的 SQL 文件，也可以导出为纯文本文件。

mysqldump 创建一个包含创建表的 CREATE TABLE 语句的 tablename.sql 文件和一个包含其数据的 tablename.txt 文件。mysqldump 导出文本文件的基本语法格式如下：

```
mysqldump -T path-u root -p dbname [tables] [OPTIONS]
```

--OPTIONS 选项

--fields-terminated-by=value

--fields-enclosed-by=value


```
--fields-optionally-enclosed-by=value
--fields-escaped-by=value
--lines-terminated-by=value
```

只有指定了 -T 参数才可以导出纯文本文件；path 表示导出数据的目录；tables 为指定要导出的表名称，如果不指定，将导出数据库 dbname 中所有的表；[OPTIONS] 为可选参数选项，这些选项需要结合 -T 选项使用。使用 OPTIONS 常见的取值有：

- --fields-terminated-by=value: 设置字段之间的分隔字符，可以为单个或多个字符，默认情况下为制表符 “\t”。
- --fields-enclosed-by=value: 设置字段的包围字符。
- --fields-optionally-enclosed-by=value: 设置字段的包围字符，只能为单个字符，只能包括 CHAR 和 VARCHAR 等字符数据类型。
- --fields-escaped-by=value: 控制如何写入或读取特殊字符，只能为单个字符，即设置转义字符，默认值为反斜线 “\”。
- --lines-terminated-by=value: 设置每行数据结尾的字符，可以为单个或多个字符，默认值为 “\n”。

提示

与 SELECT...INTO OUTFILE 语句中的 OPTIONS 各个参数设置不同，这里 OPTIONS 各个选项等号后面的 value 值不要用引号括起来。

【例 3.13】使用 mysqldump 将 test 数据库中的 person 表中的记录导出到文本文件，执行的命令如下：

```
mysqldump -T C:\test person -u root-p
```

语句执行成功，系统 C 盘目录下面将会有两个文件，分别为 person.sql 和 person.txt。person.sql 包含创建 person 表的 CREATE 语句，其内容如下：

```
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40101 SET @OLD_SQL_MODE=@SQL_MODE, SQL_MODE='' */;
/*!40111 SET @OLD_SQL_NOTES=@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `person`
--

DROP TABLE IF EXISTS `person`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `person` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` char(40) NOT NULL DEFAULT '',
```

```

`age` int(11) NOT NULL DEFAULT '0',
`info` char(50) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2011-08-19 15:02:16

```

备份文件中的信息如 3.1.1 节介绍。

person.txt 包含数据包中的数据，其内容如下：

| | | | |
|----|---------|----|------------|
| 1 | Green | 21 | Lawyer |
| 2 | Suse | 22 | dancer |
| 3 | Mary | 24 | Musician |
| 4 | Willam | 20 | sports man |
| 5 | Laura | 25 | \N |
| 6 | Evans | 27 | secretary |
| 7 | Dale | 22 | cook |
| 8 | Edison | 28 | singer |
| 9 | Harry | 21 | magician |
| 10 | Harriet | 19 | pianist |

【例 3.14】使用 mysqldump 命令将 test 数据库中的 person 表中的记录导出到文本文件，使用 FIELDS 选项，要求字段之间使用逗号“,”间隔，所有字符类型字段值用双引号括起来，定义转义字符为问号“?”，每行记录以回车换行符“\r\n”结尾，执行的命令如下：

```

C:\>mysqldump -T C:\backup test person -u root -p --fields-terminated-by=,
--fields-optionally-enclosed-by=
\" --fields-escaped-by=? --lines-terminated-by=\r\n

```

上面语句要在一行中输入，语句执行成功，系统 C:\backup 目录下面将会有两个文件，分别为 person.sql 和 person.txt。person.sql 包含创建 person 表的 CREATE 语句，其内容与前面例子中的相同，person.txt 文件的内容与上一个例子不同，显示如下：

```

1,"Green",21,"Lawyer"
2,"Suse",22,"dancer"
3,"Mary",24,"Musician"
4,"Willam",20,"sports man"

```

```

5, "Laura", 25, ?N
6, "Evans", 27, "secretary"
7, "Dale", 22, "cook"
8, "Edison", 28, "singer"
9, "Harry", 21, "magician"
10, "Harriet", 19, "pianist"

```

可以看到，只有字符类型的值被双引号括了起来，而数值类型的值没有；第 5 行记录中的 NULL 值表示为“?N”，使用问号‘?’替代了系统默认的反斜线转义字符‘\’。

3.4.3 用 MySQL 命令导出文本文件

MySQL 是一个功能丰富的工具命令，使用 MySQL 还可以在命令行模式下执行 SQL 指令，将查询结果导入到文本文件中。相比 mysqldump，MySQL 工具导出的结果可读性更强。

如果 MySQL 服务器是单独的机器，用户是在一个 client 上进行操作，用户要把数据结果导入到 client 机器上。可以使用 mysql -e 语句。

使用 MySQL 导出数据文本文件语句的基本格式如下：

```
mysql -u root -p --execute= "SELECT 语句" dbname > filename.txt
```

该命令使用--execute 选项，表示执行该选项后面的语句并退出，后面的语句必须用双引号括起来，dbname 为要导出的数据库名称；导出的文件中不同列之间使用制表符分隔，第 1 行包含了各个字段的名称。

【例 3.15】使用 MySQL 语句，导出 test 数据库中 person 表中的记录到文本文件，输入语句如下：

```
mysql -u root -p --execute="SELECT * FROM person;" test > C:\person3.txt
```

语句执行完毕之后，系统 C 盘目录下面将会有名称为 person3.txt 的文本文件，其内容如下：

| id | name | age | info |
|----|---------|-----|------------|
| 1 | Green | 21 | Lawyer |
| 2 | Suse | 22 | dancer |
| 3 | Mary | 24 | Musician |
| 4 | Willam | 20 | sports man |
| 5 | Laura | 25 | NULL |
| 6 | Evans | 27 | secretary |
| 7 | Dale | 22 | cook |
| 8 | Edison | 28 | singer |
| 9 | Harry | 21 | magician |
| 10 | Harriet | 19 | pianist |

可以看到，person3.txt 文件中包含了每个字段的名称和各条记录，该显示格式与 MySQL 命令行下 SELECT 查询结果显示相同。

使用 MySQL 命令还可以指定查询结果的显示格式，如果某行记录字段很多，可能一行不

能完全显示，可以使用--vertical 参数，将每条记录分为多行显示。

【例 3.16】使用 mysql 命令导出 test 数据库中 person 表中的记录到文本文件，使用--vertical 参数显示结果，输入语句如下：

```
mysql -u root -p --vertical --execute="SELECT * FROM person;" test >
C:\person4.txt
```

语句执行之后，C:\person4.txt 文件中的内容如下：

```
***** 1. row *****
id: 1
name: Green
age: 21
info: Lawyer
***** 2. row *****
id: 2
name: Suse
age: 22
info: dancer
***** 3. row *****
id: 3
name: Mary
age: 24
info: Musician
***** 4. row *****
id: 4
name: Willam
age: 20
info: sports man
***** 5. row *****
id: 5
name: Laura
age: 25
info: NULL
***** 6. row *****
id: 6
name: Evans
age: 27
info: secretary
***** 7. row *****
id: 7
name: Dale
age: 22
info: cook
***** 8. row *****
id: 8
name: Edison
age: 28
info: singer
***** 9. row *****
id: 9
name: Harry
```

```
age: 21
info: magician
***** 10. row *****
id: 10
name: Harriet
age: 19
info: pianist
```

可以看到，SELECT 的查询结果导出到文本文件之后，显示格式发生了变化，如果 person 表中记录内容很长，这样显示将会更加容易阅读。

MySQL 可以将查询结果导出到 html 文件中，使用--html 选项即可。

【例 3.17】使用 MySQL 命令导出 test 数据库中 person 表中的记录到 html 文件，输入语句如下：

```
mysql -u root -p --html --execute="SELECT * FROM person;" test > C:\person5.html
```

语句执行成功，将在 C 盘创建文件 person5.html，该文件在浏览器中显示如图 3-1 所示。

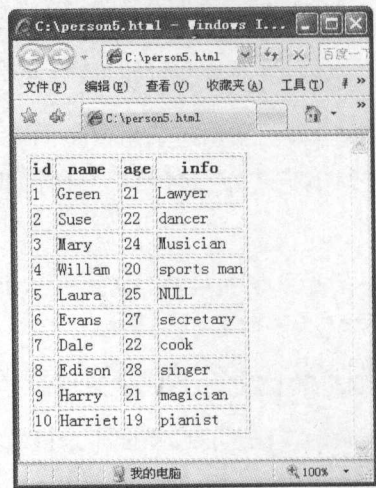


图 3-1 使用 mysql 导出数据到 html 文件

如果要表数据导出到 xml 文件中，可使用--xml 选项。

【例 3.18】使用 MySQL 命令导出 test 数据库中 person 表中的记录到 xml 文件，输入语句如下：

```
mysql -u root -p --xml --execute="SELECT * FROM person;" test > C:\person6.xml
```

语句执行成功，将在 C 盘创建文件 person6.xml，该文件在浏览器中显示如图 3-2 所示。

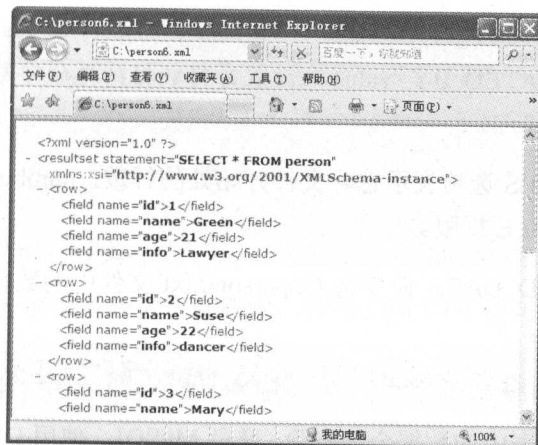


图 3-2 使用 mysql 导出数据到 xml 文件

3.4.4 使用 LOAD DATA INFILE 方式导入文本文件

MySQL 允许将数据导出到外部文件，也可以从外部文件导入数据。MySQL 提供了一些导入数据的工具，这些工具有 LOAD DATA 语句、source 命令和 mysql 命令。LOAD DATA INFILE 语句用于高速地从文本文件中读取行，并装入一个表中。文件名称必须为文字字符串。本节将介绍 LOAD DATA 语句的用法。

LOAD DATA 语句的基本格式如下：

```
LOAD DATA INFILE 'filename.txt' INTO TABLE tablename [OPTIONS] [IGNORE number
LINES]
```

-- OPTIONS 选项

FIELDS TERMINATED BY 'value'

FIELDS [OPTIONALLY] ENCLOSED BY 'value'

FIELDS ESCAPED BY 'value'

LINES STARTING BY 'value'

LINES TERMINATED BY 'value'

可以看到 LOAD DATA 语句中，关键字 INFILE 后面的 filename 文件为导入数据的来源；tablename 表示待导入的数据表名称；[OPTIONS] 为可选参数选项，OPTIONS 部分的语法包括 FIELDS 和 LINES 子句，其可能的取值有：

- FIELDS TERMINATED BY 'value': 设置字段之间的分隔字符，可以为单个或多个字符，默认情况下为制表符“\t”。
- FIELDS [OPTIONALLY] ENCLOSED BY 'value': 设置字段的包围字符，只能为单个字符。如果使用了 OPTIONALLY，则只有 CHAR 和 VARCHAR 等字符数据字段被包括。
- FIELDS ESCAPED BY 'value': 控制如何写入或读取特殊字符，只能为单个字符，即设置转义字符，默认值为“\”。
- LINES STARTING BY 'value': 设置每行数据开头的字符，可以为单个或多个字符，

默认情况下不使用任何字符。

- **LINES TERMINATED BY 'value':** 设置每行数据结尾的字符，可以为单个或多个字符，默认值为“\n”。

IGNORE number LINES 选项表示忽略文件开始处的行数，**number** 表示忽略的行数。执行 **LOAD DATA** 语句需要 **FILE** 权限。

【例 3.19】使用 **LOAD DATA** 命令将 **C:\person0.txt** 文件中的数据导入到 **test** 数据库中的 **person** 表，输入语句如下：

```
LOAD DATA INFILE 'C:\person0.txt' INTO TABLE test.person;
```

还原之前，将 **person** 表中的数据全部删除，登录 **MySQL**，使用 **DELETE** 语句，语句如下：

```
mysql> USE test;
Database changed;
mysql> DELETE FROM person;
Query OK, 10 rows affected (0.00 sec)
```

从 **person0.txt** 文件中还原数据，语句如下：

```
mysql> LOAD DATA INFILE 'C:\person0.txt' INTO TABLE test.person;
Query OK, 10 rows affected (0.00 sec)
Records: 10 Deleted: 0 Skipped: 0 Warnings: 0
```

```
mysql> SELECT * FROM person;
+----+-----+-----+-----+
| id | name  | age  | info          |
+----+-----+-----+-----+
| 1  | Green | 21   | Lawyer        |
| 2  | Suse  | 22   | dancer        |
| 3  | Mary  | 24   | Musician      |
| 4  | Willam | 20   | sports man    |
| 5  | Laura | 25   | NULL          |
| 6  | Evans | 27   | secretary     |
| 7  | Dale  | 22   | cook          |
| 8  | Edison | 28   | singer        |
| 9  | Harry | 21   | magician      |
| 10 | Harriet | 19   | pianist       |
+----+-----+-----+-----+
10 rows in set (0.00 sec)
```

可以看到，语句执行成功之后，原来的数据重新恢复到了 **person** 表中。

【例 3.20】使用 **LOAD DATA** 命令将 **C:\person1.txt** 文件中的数据导入到 **test** 数据库中的 **person** 表，使用 **FIELDS** 选项和 **LINES** 选项，要求字段之间使用逗号 ‘,’ 间隔，所有字段值用双引号括起来，定义转义字符为单引号 “\”，输入语句如下：

```
LOAD DATA INFILE 'C:\person1.txt' INTO TABLE test.person
FIELDS
TERMINATED BY ','
ENCLOSED BY '\"'
ESCAPED BY '\\'
LINES
TERMINATED BY '\r\n';
```

还原之前，将 **person** 表中的数据全部删除，使用 **DELETE** 语句，执行过程如下：

```
mysql> DELETE FROM person;
Query OK, 10 rows affected (0.00 sec)
```

从 **person1.txt** 文件中还原数据，执行过程如下：

```
mysql> LOAD DATA INFILE 'C:\person1.txt' INTO TABLE test.person
-> FIELDS
-> TERMINATED BY ','
-> ENCLOSED BY '\"'
-> ESCAPED BY '\\'
-> LINES
-> TERMINATED BY '\r\n';
Query OK, 10 rows affected (0.00 sec)
Records: 10 Deleted: 0 Skipped: 0 Warnings: 0
```

语句执行成功，使用 **SELECT** 语句查看 **person** 表中的记录，结果与前一个例子相同。

3.4.5 使用 **mysqlimport** 命令导入文本文件

使用 **mysqlimport** 命令可以导入文本文件，并且不需要登录 MySQL 客户端。**mysqlimport** 命令提供许多与 **LOAD DATA INFILE** 语句相同的功能，大多数选项直接对应 **LOAD DATA INFILE** 子句。使用 **mysqlimport** 语句需要指定所需的选项、导入的数据库名称以及导入的数据文件的路径和名称。**mysqlimport** 命令的基本语法格式如下：

```
mysqlimport -u root -p dbname filename.txt [OPTIONS]
```

```
--OPTIONS 选项
--fields-terminated-by=value
--fields-enclosed-by=value
--fields-optionally-enclosed-by=value
--fields-escaped-by=value
--lines-terminated-by=value
--ignore-lines=n
```

dbname 为导入的表所在的数据库名称。注意，**mysqlimport** 命令不指定导入数据库的表名称，数据表的名称由导入文件名称确定，即文件名作为表名，导入数据之前该表必须存在。**[OPTIONS]** 为可选参数选项，其常见的取值有：

- `--fields-terminated-by= 'value'`: 设置字段之间的分隔字符, 可以为单个或多个字符, 默认情况下为制表符 “\t”。
- `--fields-enclosed-by= 'value'`: 设置字段的包围字符。
- `--fields-optionally-enclosed-by= 'value'`: 设置字段的包围字符, 只能为单个字符, 包括 CHAR 和 VARCHAR 等字符数据字段。
- `--fields-escaped-by= 'value'`: 控制如何写入或读取特殊字符, 只能为单个字符, 即设置转义字符, 默认值为反斜线 “\”。
- `--lines-terminated-by= 'value'`: 设置每行数据结尾的字符, 可以为单个或多个字符, 默认值为 “\n”。
- `--ignore-lines=n`: 忽视数据文件的前 n 行。

【例 3.21】使用 `mysqlimport` 命令将 C:\backup 目录下的 `person.txt` 文件内容导入到 test 数据库中, 字段之间使用逗号 “,” 间隔, 字符类型字段值用双引号括起来, 将转义字符定义为问号 “?”, 每行记录以回车换行符 “\r\n” 结尾, 执行的命令如下:

```
C:\>mysqlimport -u root -p test C:\backup/person.txt --fields-terminated-by=,
--fields-optionally-enclosed-
by=\" --fields-escaped-by=? --lines-terminated-by=\r\n
```

上面语句要在一行中输入, 语句执行成功, 将把 `person.txt` 中的数据导入到数据库。

除了前面介绍的几个选项之外, `mysqlimport` 支持许多选项, 常见的选项有:

- `--columns=column_list, -c column_list`: 该选项采用逗号分隔的列名作为其值。列名的顺序指示如何匹配数据文件列和表列。
- `--compress, -C`: 压缩在客户端和服务端之间发送的所有信息 (如果二者均支持压缩)。
- `-d, --delete`: 导入文本文件前清空表。
- `--force, -f`: 忽视错误。例如, 如果某个文本文件的表不存在, 继续处理其他文件。不使用 `--force`, 如果表不存在, 则 `mysqlimport` 退出。
- `--host=host_name, -h host_name`: 将数据导入给定主机上的 MySQL 服务器。默认主机是 localhost。
- `--ignore, -i`: 参见 `--replace` 选项的描述。
- `--ignore-lines=n`: 忽视数据文件的前 n 行。
- `--local, -L`: 从本地客户端读入输入文件。
- `--lock-tables, -l`: 处理文本文件前锁定所有表以便写入。这样可以确保所有表在服务器上保持同步。
- `--password[=password], -p[password]`: 当连接服务器时使用的密码。如果使用短选项形式 (`-p`), 选项和密码之间不能有空格。如果在命令行中 `--password` 或 `-p` 选项后面没有密码值, 则提示输入一个密码。
- `--port=port_num, -P port_num`: 用于连接的 TCP/IP 端口号。
- `--protocol={TCP | SOCKET | PIPE | MEMORY}`: 使用的连接协议。

- `--replace`, `-r` `--replace` 和 `--ignore` 选项控制复制唯一键值已有记录的输入记录的处理。如果指定 `--replace`, 新行替换有相同的唯一键值的已有行; 如果指定 `--ignore`, 复制已有的唯一键值的输入行被跳过; 如果不指定这两个选项, 当发现一个复制键值时会出现一个错误, 并且忽视文本文件的剩余部分。
- `--silent`, `-s`: 沉默模式。只有出现错误时才输出信息。
- `--user=user_name`, `-u user_name`: 当连接服务器时 MySQL 使用的用户名。
- `--verbose`, `-v`: 冗长模式。打印出程序操作的详细信息。
- `--version`, `-V`: 显示版本信息并退出。

3.5 综合实例——数据的备份与恢复

备份有助于保护数据库, 通过备份可以完整保存 MySQL 中各个数据库的特定状态。在系统出现故障、数据丢失或者不合理操作对数据库造成灾难时, 可以通过还原恢复数据库中的数据。作为 MySQL 的管理人员, 应该定期地备份所有活动的数据库, 以免发生数据丢失。因此, 无论怎样强调数据库的备份工作都不过分。本章综合实例将向读者提供数据库备份与还原的方法与过程。

1. 案例目的

按照操作过程完成对 test 数据库的备份和还原。

- 步骤 01** 使用 `mysqldump` 命令将 `suppliers` 表备份到文件 `C:\bktestdir\suppliers_bk.sql`。
- 步骤 02** 使用 `mysql` 命令还原 `suppliers` 表到 test 数据库中。
- 步骤 03** 使用 `SELECT... INTO OUTFILE` 语句导出 `suppliers` 表中的记录, 导出文件位于目录 `C:\bktestdir` 下, 名称为 `suppliers_out.txt`。
- 步骤 04** 使用 `LOAD DATA INFILE` 语句导入 `suppliers_out.txt` 数据到 `suppliers` 表。
- 步骤 05** 使用 `mysqldump` 命令将 `suppliers` 表中的记录导出到文件 `C:\bktestdir\suppliers_html.html`。

2. 案例操作过程

- 步骤 01** 使用 `mysqldump` 命令将 `suppliers` 表备份到文件 `C:\bktestdir\suppliers_bk.sql`。

首先创建系统目录, 在系统 C 盘下面新建文件夹 `bktestdir`, 然后打开命令行窗口, 输入语句如下:

```
C:\>mysqldump -u root -p test suppliers > C:\bktestdir\suppliers_bk.sql
Enter password: **
```

语句执行完毕, 打开目录 `C:\bktestdir`, 可以看到已经创建好的备份文件 `suppliers_bk.sql`,

内容如下:

```
-- MySQL dump 10.13 Distrib 5.6.10, for Win32 (x86)
--
-- Host: localhost Database: test
--
-- Server version 5.6.10

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `suppliers`
--

DROP TABLE IF EXISTS `suppliers`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `suppliers` (
  `s_id` int(11) NOT NULL AUTO_INCREMENT,
  `s_name` char(50) NOT NULL,
  `s_city` char(50) DEFAULT NULL,
  `s_zip` char(10) DEFAULT NULL,
  `s_call` char(50) NOT NULL,
  PRIMARY KEY (`s_id`)
) ENGINE=InnoDB AUTO_INCREMENT=108 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `suppliers`
--

LOCK TABLES `suppliers` WRITE;
/*!40000 ALTER TABLE `suppliers` DISABLE KEYS */;
INSERT INTO `suppliers` VALUES (101,'FastFruit Inc.','Tianjin','463400','48075'),
(102,'LT Supplies','Chongqing','100023','44333'),(103,'ACME','Shanghai','100024',
'90046'),(104,'FNK Inc.','Zhongshan','212021','11111'), (105,'Good Set',
```

```
'Taiyuan', '230009', '22222'), (106, 'Just Eat Ours', 'Beijing', '010', '45678'),
(107, 'DK Inc.', 'Qingdao', '230009', '33332');
/*!40000 ALTER TABLE `suppliers` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2011-08-20 15:07:44
```

步骤 02 使用 mysql 命令将备份文件 suppliers_bk.sql 中的数据还原 suppliers 表。

为了验证还原之后数据的正确性, 删除 suppliers 表中的所有记录, 登录 MySQL, 输入语句:

```
mysql> USE test;
Database changed
mysql> DELETE FROM suppliers;
Query OK, 7 rows affected (0.00 sec)
```

此时, suppliers 表中不再有任何数据记录, 在 MySQL 命令行输入还原语句如下:

```
mysql> source C:\bktestdir\suppliers_bk.sql;
```

语句执行过程中会出现多行提示信息, 执行成功之后使用 SELECT 语句查询 suppliers 表内容如下:

```
mysql> SELECT * FROM suppliers;
+-----+-----+-----+-----+-----+
| s_id | s_name      | s_city   | s_zip | s_call |
+-----+-----+-----+-----+-----+
| 101  | FastFruit Inc. | Tianjin  | 463400 | 48075 |
| 102  | LT Supplies   | Chongqing | 100023 | 44333 |
| 103  | ACME          | Shanghai | 100024 | 90046 |
| 104  | FNK Inc.      | Zhongshan | 212021 | 11111 |
| 105  | Good Set      | Taiyuan  | 230009 | 22222 |
| 106  | Just Eat Ours | Beijing  | 010    | 45678 |
| 107  | DK Inc.       | Qingdao  | 230009 | 33332 |
+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

由查询结果可以看到, 还原操作成功。

步骤 03 使用 SELECT... INTO OUTFILE 语句导出 suppliers 表中的记录，导出文件位于目录 C:\bktestdir 下，名称为 suppliers_out.txt。

执行过程如下：

```
mysql> SELECT * FROM test.suppliers INTO OUTFILE
"C:\bktestdir\suppliers_out.txt"
-> FIELDS
-> TERMINATED BY ','
-> ENCLOSED BY '\"'
-> LINES
-> STARTING BY '<'
-> TERMINATED BY '>\r\n';
Query OK, 7 rows affected (0.00 sec)
```

TERMINATED BY ',' 指定不同字段之间使用逗号分隔开；ENCLOSED BY '\"' 指定字段值使用双引号包括；STARTING BY '<' 指定每行记录以左箭头符号开始；TERMINATED BY '>\r\n' 指定每行记录以右箭头符号和回车换行符结束。语句执行完毕，打开目录 C:\bktestdir，可以看到已经创建好的导出文件 suppliers_out.txt，内容如下：

```
<"101","FastFruit Inc.,""Tianjin","463400","48075">
<"102","LT Supplies","Chongqing","100023","44333">
<"103","ACME","Shanghai","100024","90046">
<"104","FNK Inc.,""Zhongshan","212021","11111">
<"105","Good Set","Taiyuan","230009","22222">
<"106","Just Eat Ours","Beijing","010","45678">
<"107","DK Inc.,""Qingdao","230009","33332">
```

步骤 04 使用 LOAD DATA INFILE 语句导入 suppliers_out.txt 数据到 suppliers 表。

首先使用 DELETE 语句删除 suppliers 表中的所有记录，然后输入导入语句：

```
mysql> LOAD DATA INFILE 'C:\bktestdir\suppliers_out.txt' INTO TABLE
test.suppliers
-> FIELDS
-> TERMINATED BY ','
-> ENCLOSED BY '\"'
-> LINES
-> STARTING BY '<'
-> TERMINATED BY '>\r\n';
Query OK, 7 rows affected (0.00 sec)
Records: 7 Deleted: 0 Skipped: 0 Warnings: 0
```

语句执行之后，suppliers_out.txt 文件中的数据将导入 suppliers 表中，由于导出 txt 文件时指定了一些特殊字符，因此还原语句中也要指定这些字符，已确保还原后数据的完整性和正确性。

步骤 05 使用 MySQL 命令将 suppliers 表中的记录导出到文件 C:\bktestdir\suppliers_html.html。

导出表数据到 html 文件，使用 MySQL 命令时需要指定--html 选项，在 Windows 命令行窗口输入导出语句如下：

```
mysql -u root -p --html --execute="SELECT * FROM suppliers;" test >
C:\bktestdir\suppliers_html.html
```

语句执行完毕，打开目录 C:\bktestdir，可以看到已经创建好的导出文件 suppliers_html.html，读者可以使用浏览器打开该文件，在浏览器中显示格式和内容如表 3-1 所示。

表 3-1 浏览器中显示导出文件的内容

| s_id | s_name | s_city | s_zip | s_call |
|------|----------------|-----------|--------|--------|
| 101 | FastFruit Inc. | Tianjin | 463400 | 48075 |
| 102 | LT Supplies | Chongqing | 100023 | 44333 |
| 103 | ACME | Shanghai | 100024 | 90046 |
| 104 | FNK Inc. | Zhongshan | 212021 | 11111 |
| 105 | Good Set | Taiyuan | 230009 | 22222 |
| 106 | Just Eat Ours | Beijing | 010 | 45678 |
| 107 | DK Inc. | Qingdao | 230009 | 33332 |

3.6 小结

本章主要介绍了 MySQL 数据库的备份和恢复方法。包括备份单表、多表和多个数据库等实际的工作需求。针对不同的备份文件，还讲述了不同的还原方法。另外讲述了数据库的迁移操作，包括相同版本和不同版本的迁移，不同数据库之间迁移等，最后讲述了数据表的导入和导出方法。通过本章的学习，读者可以根据实际的工作要求，选择适合的备份和还原的方法。

第 4 章

◀ MySQL 的高级特性 ▶

本章主要讲解 MySQL 的一些高级特性，其中包括 MySQL 查询缓存，查询缓存会存储一个 SELECT 查询的文本与被传送到客户端的相应结果。如果执行相同的一个 SQL 语句，MySQL 数据库会将数据缓存起来以供下次直接使用，MySQL 数据库以此优化查询缓存来提高缓存命中率。

MySQL 5.1 及高版本支持分区表 (partitioned table)，分区表的使用大大增加了 MySQL 执行效率。另外本章还讲解到 MySQL 数据库事务，其中包括分布式事务的原理和语法。MySQL 分布式事务涉及多个事务性的活动，本章介绍分布式事务使用的同时也涉及 MySQL 分布式事务技术存在的一些漏洞。

4.1 MySQL 查询缓存

MySQL 服务器有一个重要特征是查询缓存。缓存机制简单地讲就是缓存 SQL 语句和查询的结果，如果运行相同的 SQL 语句，服务器会直接从缓存中取到结果，而不需要再去解析和执行 SQL 语句。查询缓存会存储最新数据，而不会返回过期数据。当数据被修改后，在查询缓存中的任何相关数据均被清除。对于频繁更新的表，查询缓存是不适合的；而对于一些不经常改变数据且有大量相同 SQL 查询的表，查询缓存会提高很大的性能。

4.1.1 认识查询缓存

MySQL 数据库设置了查询缓存后，当服务器接收到一个和之前同样的查询时，会从查询缓存中检索查询结果，而不是直接分析并检索查询。

在 MySQL 数据库中，使用查询缓存功能的具体操作步骤如下。

步骤 01 设置 query_cache_type 为 ON，命令如下：

```
mysql> set session query_cache_type=ON;  
Query OK, 0 rows affected (0.01 sec)
```

步骤 02 查看查询缓存功能是否被开启，命令如下：

```
mysql> select @@query_cache_type;  
+-----+  
| @@query_cache_type |
```



```

+-----+
| ON |
+-----+
1 row in set (0.00 sec)

```

从结果可以看出，查询缓存功能已经被开启。

提示

如果要禁用查询缓存功能，直接执行命令如下：

```
mysql> set session query_cache_type=OFF;
```

步骤 03 查看系统变量 `have_query_cache` 是否为 'YES'，该参数表示 MySQL 的查询缓存是否可用，查看命令如下。

```

mysql> show variables like 'have_query_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
+-----+-----+
1 row in set (0.00 sec)

```

步骤 04 查询系统变量 `query_cache_size` 的大小，该参数表示数据库分配给查询缓存的内存大小，如果该参数的值设置为 0，那么查询缓存功能将不起任何作用。

```

mysql> select @@global.query_cache_size;
+-----+
| @@global.query_cache_size |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)

```

从结果可知，系统默认的 `query_cache_size` 参数值是 0。

步骤 05 设置系统变量 `query_cache_size` 的大小，命令如下：

```

mysql> set @@global.query_cache_size=1000000;
Query OK, 0 rows affected (0.00 sec)

```

步骤 06 查询系统变量 `query_cache_size` 设置后的大小，命令如下：

```

mysql> select @@global.query_cache_size;
+-----+
| @@global.query_cache_size |
+-----+
| 1000000 |
+-----+
1 row in set (0.00 sec)

```

步骤 07 如果要将该参数永久修改，需要修改 `/etc/my.cnf` 配置文件，添加该参数的选项，添加如下：

```
[mysqld]
port = 3306
query_cache_size = 1000000
...
```

步骤 08 如果查询结果很大，也可能缓存不了，需要设置 `query_cache_limit` 参数的值，该参数用来设置查询缓存的最大值，该值默认是 1MB。

查询该参数的值的命令如下：

```
mysql> select @@global.query_cache_limit;
+-----+
| @@global.query_cache_limit |
+-----+
| 1000000 |
+-----+
1 row in set (0.00 sec)
```

步骤 09 设置 `query_cache_limit` 参数值的大小，命令如下：

```
mysql> set @@global.query_cache_limit=2000000;
Query OK, 0 rows affected (0.00 sec)
```

步骤 10 如果需要将该参数永久修改，需要修改 `/etc/my.cnf` 配置文件，添加该参数的选项，添加如下：

```
[mysqld]
port = 3306
query_cache_size=1000000
query_cache_limit=2000000
...
```

通过以上步骤的设置，MySQL 数据库已经成功地开启查询缓存功能。在实际工作中，需要关注查询缓存的使用效率和性能，可以使用 `SHOW VARIABLES` 命令查询缓存的相关参数，命令如下。

```
mysql> show variables like '%query_cache%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_query_cache | YES |
| query_cache_limit | 1000000 |
| query_cache_min_res_unit | 4096 |
| query_cache_size | 0 |
| query_cache_type | ON |
| query_cache_wlock_invalidate | OFF |
+-----+-----+
6 rows in set (0.00 sec)
```

下面具体介绍查询缓存功能相关参数的含义。

- `have_query_cache` 用来设置是否支持查询缓存区，“YES”表示支持查询缓存区。
- `query_cache_limit` 用来设置 MySQL 可以缓存的最大结果集，大于此值的结果集不会

被缓存, 该参数默认值是 1MB。

- `query_cache_min_res_unit` 用来设置分配内存块的最小体积。每次给查询缓存结果分配内存的大小, 默认分配 4096 个字节。如果此值较小, 那么会节省内存, 但是这样会使系统频繁分配内存块。
- `query_cache_size` 用来设置查询缓存使用的总内存字节数, 必须是 1024 字节的倍数。
- `query_cache_type` 用来设置是否启用查询缓存。如果设置为 OFF, 表示不进行缓存; 如果设置为 ON, 表示除了 SQL_NO_CACHE 的查询以外, 缓存所有的结果; 如果设置为 DEMAND, 表示仅缓存 SQL_CACHE 的查询。
- `query_cache_wlock_invalidate` 用来设置是否允许在其他连接处于 lock 状态时, 使用缓存结果, 默认是 OFF, 不会影响大部分应用。在默认情况下, 一个查询中使用的表即使被 LOCK TABLES 命令锁住了, 查询也能被缓存下来。可以通过设置该参数来关闭这个功能。

下面通过一个简单的例子来了解查询缓存的过程。

步骤 01 设置缓存内存大小为 10240 字节, 开启查询缓存功能, 命令如下。

```
mysql> set global query_cache_size=10240;
Query OK, 0 rows affected (0.00 sec)
mysql> set session query_cache_type=ON;
Query OK, 0 rows affected (0.01 sec)
```

步骤 02 查询 t 表中总共记录的条数, 命令如下。

```
mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| rep_t1          |
| rep_t2          |
| t               |
+-----+
3 rows in set (0.00 sec)
mysql> use test;
Database changed
mysql> select count(*) from t;
+-----+
| count(*) |
+-----+
|      64 |
+-----+
1 row in set (0.05 sec)
```

步骤 03 查询在缓存中命中的累计次数, 命令如下:

```
mysql> show status like 'Qcache_hits';
+-----+-----+
```



```

| Variable_name | Value |
+-----+-----+
| Qcache_hits   | 0     |
+-----+-----+
1 row in set (0.00 sec)

```

从结果可知，第一次查询发现查询缓存累计命中的数是 0。

步骤 04 再次查询 t 表的总的记录数据，然后查询缓存累计命中数，命令如下：

```

mysql> select count(*) from t;
+-----+
| count(*) |
+-----+
|      64 |
+-----+
1 row in set (0.00 sec)
mysql> show status like 'Qcache_hits';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_hits   | 1     |
+-----+-----+
1 row in set (0.00 sec)

```

从结果可知，第二次查询后发现该缓存累计命中数已经发生了变化，此时查询出参数 Qcache_hits 的值是 1，表示查询直接从缓存中获取结果，不需要再去解析 SQL 语句。

步骤 05 连续两次查询 t 表的总的记录数据，然后再查询缓存累计命中数，此时会发现缓存累计命中数已经变成了 3，本次查询也是直接从缓存中取到结果，执行命令如下。

```

mysql> select count(*) from t;
+-----+
| count(*) |
+-----+
|      64 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from t;
+-----+
| count(*) |
+-----+
|      64 |
+-----+
1 row in set (0.00 sec)

mysql> show status like 'Qcache_hits';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_hits   | 3     |
+-----+-----+

```

```
1 row in set (0.00 sec)
```

步骤 06 增加了一条记录到 t 表，插入数据后，跟该表所有相关的查询缓存就会被清空掉，然后重新查询 t 表的总的记录，此时发现缓存累计命中数没有发生变化，说明本次查询没有直接从缓存中取到数据，执行命令如下。

```
mysql> insert into t values(2);
Query OK, 1 row affected (0.00 sec)
mysql> select count(*) from t;
+-----+
| count(*) |
+-----+
|      65 |
+-----+
1 row in set (0.00 sec)

mysql> show status like 'Qcache_hits';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_hits   | 3     |
+-----+-----+
1 row in set (0.00 sec)
```

步骤 07 重新查询 t 表的总的记录，此时发现缓存累计命中数发生了变化，说明缓存继续起作用了，执行命令如下。

```
mysql> select count(*) from t;
+-----+
| count(*) |
+-----+
|      65 |
+-----+
1 row in set (0.00 sec)

mysql> show status like 'Qcache_hits';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_hits   | 4     |
+-----+-----+
1 row in set (0.00 sec)
```

通过上面的例子不难发现，查询缓存适合更新不频繁的表，当表更改后，查询缓存值的相关条目被清空。

4.1.2 监控和维护查询缓存

在日常工作中，经常使用以下命令监控和维护查询缓存。

(1) flush query cache: 该命令用于整理查询缓存，以便更好地利用查询缓存的内存，这

个命令不会从缓存中移除任何查询结果。命令运行如下：

```
mysql> flush query cache;
Query OK, 0 rows affected (0.00 sec)
```

(2) reset query cache: 该命令用于移除查询缓存中所有的查询结果。命令运行如下：

```
mysql> reset query cache;
Query OK, 0 rows affected (0.00 sec)
```

(3) show status like 'Qcache%': 该命令可以监视查询缓存的使用状况，可以计算出缓存命中率。命令运行如下：

```
mysql> show status like 'Qcache%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_free_blocks | 4984 |
| Qcache_free_memory | 30097400 |
| Qcache_hits | 24 |
| Qcache_inserts | 4342 |
| Qcache_lowmem_prunes | 41224 |
| Qcache_not_cached | 2654 |
| Qcache_queries_in_cache | 20527 |
| Qcache_total_blocks | 46362 |
+-----+-----+
8 rows in set (0.00 sec)
```

结果中的性能监控参数含义如表 4-1 所示。

表 4-1 查询缓存的性能监控参数含义

| 变量 | 含义 |
|-------------------------|---------------------|
| Qcache queries in cache | 在缓存中已注册的查询数目 |
| Qcache inserts | 被加入到缓存中的查询数目 |
| Qcache hits | 缓存采样数的数目 |
| Qcache lowmem prunes | 因为缺少内存而被从缓存中删除的查询数目 |
| Qcache not cached | 没有被缓存的查询数目 |
| Qcache free memory | 查询缓存的空闲内存总数 |
| Qcache free blocks | 查询缓存中的空闲内存块的数目 |
| Qcache total blocks | 查询缓存中的块的总数目 |

如果空闲内存块是总内存块的一半左右，则表示存在严重的内存碎片。通常使用 flush query cache 命令整理碎片，然后采用 reset query cache 命令清理查询缓存。

如果碎片很少，但是缓存命中率很低，则说明设置的缓存内存空间过小，服务器频繁删除旧的查询缓存，腾出空间，以保存新的查询缓存，此时，参数 Qcache_lowmeme_preunes 状态值将会增加，如果此值增加过快，可能是有以下原因造成：

- 如果存在大量空闲块，则是因为碎片的存在而引起的。
- 空闲内存块较少，可以适当地增加缓存大小。

4.1.3 如何检查缓存命中率

MySQL 检查缓存命中率的方式十分简单快捷。缓存就是一个查找表 (Lookup Table)。查找的键就是查询文本、当前数据库、客户端协议的版本,以及其他少数会影响实际查询结果的因素的哈希值。

下面主要学习 MySQL 数据库中缓存的管理技巧,以及如何合理配置 MySQL 数据库缓存,提高缓存命中率。

首先,在配置数据库客户端或者是第三方工具与服务器连接时,应该保证数据库客户端的字符集跟服务器的字符集保持一致。在实际工作中,经常发现客户端配置的字符集和服务器字符集兼容没有完全一致,即使此时客户端没有出现乱码情况,查询数据可能就因为字符集不同的原因而没有被数据库缓存起来。

其次,为了提高数据库缓存的命中率,应该在客户端和服务端采用一样的 SQL 语句。从数据库缓存的角度考虑,数据库查询 SQL 的语句是不区分大小写的,比如第一个查询语句采用大写语句,第二个查询语句采用小写语句,但对于缓存来讲,大小写不同的 SQL 语句会被当作不同的查询语句。

查询缓存不会存储不确定结果的查询,任何一个包含不确定函数(比如 NOW() 或 CURRENT_DATE())的查询不会被缓存。同样地, CURRENT_USER() 或 CONNECTION_ID() 这些由不同用户执行,将会产生不同的结果的查询也不会被缓存。实际上,查询缓存不会缓存引用了用户自定义函数、存储函数、用户自定义变量、临时表的查询。

查询缓存只是发生在服务器第一次接收到 SQL 查询语句,然后把查询结果缓存起来,对于查询中的子查询、视图查询和存储过程查询都不能缓存结果,对于预存储语句同样也不能使用缓存。

使用查询缓存有利也有弊。一方面,查询缓存可以使查询变得更加高效,改善了 MySQL 服务器的性能;另一方面,查询缓存本身也需要消耗系统 IO 资源,所以说,查询缓存也增加了服务器额外的开销,主要体现在以下几个方面。

- MySQL 服务器在进行查询之前首先会检测查询缓存是否存在相同的查询条目。
- MySQL 服务器在进行查询操作时,如果缓存中没有相同的查询条目,会将查询的结果缓存到查询缓存,这个过程也需要开销系统资源。
- 如果数据库表发生增加操作,MySQL 服务器查询缓存中相对应的查询结果将会无效,这时同样需要消耗系统资源。

除了注意以上问题可以提高查询缓存的命中率外,还可以通过分区表提高缓存的命中率。通常我们会遇到这样的问题,对于某张表某个时间段内的数据更新比较频繁,其他时间段查询和更新比较多,一旦数据表数据执行更新操作,那么查询缓存中的信息将会清空,查询缓存的命中率不会很高。此时,可以考虑采用分区表,把某个时间段的数据存放在一个单独的分区表中,这样可以提高服务器的查询缓存的命中率。

4.1.4 优化查询缓存

MySQL 查询缓存优化方案的大致步骤如图 4-1 所示。

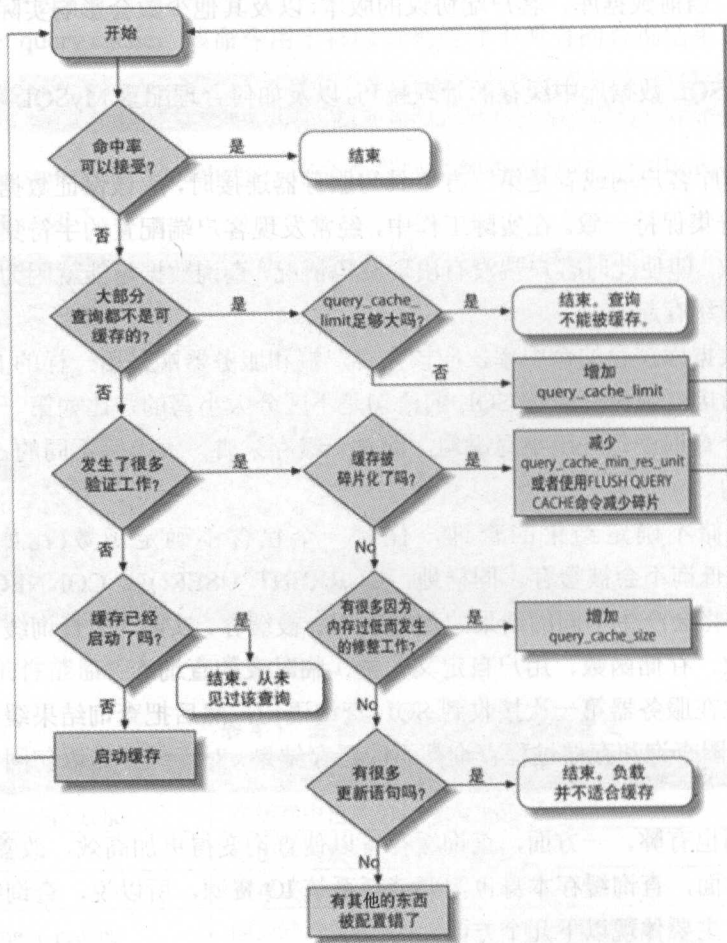


图 4-1 MySQL 优化查询缓存的大致步骤

优化查询缓存通常需要注意如下几点。

- (1) 在数据库设计的时候，尽量不要使用一张比较大的表，可以使用很多小的表，这样可以提高数据查询缓存的效率。
- (2) 在对数据库进行写操作的时候，尽量一次性写入。因为如果逐个写入操作，每次写操作都会让数据库缓存功能失效或清理缓存数据，此时服务器可能会挂起相当长时间。
- (3) 尽量不要在数据库或者表的基础上控制查询缓存，可以采用 SQL_CACHE 和 SQL_NO_CACHE 来决定是否使用缓存查询。
- (4) 可以基于某个连接来运行或禁止缓存，可以通过用适当的值设定 query_cache_size 来开启或关闭对某个连接的缓存。
- (5) 对于包含很多写入任务的应用程序，关闭查询缓存功能可以改进服务器性能。

(6) 禁用查询缓存的时候可以将 `query_cache_size` 参数设置为 0, 这样就不会消耗任何内存。

(7) 如果想少数查询使用缓存, 而多数查询都不使用查询缓存, 此时可以将全局变量 `query_cache_type` 设置为 `DEMAND`, 然后可以在想使用缓存功能的语句后面加上 `SQL_CACHE`, 不想使用缓存查询的语句后面可以加上 `SQL_NO_CACHE`, 这样可以通过语句来控制查询缓存, 提高缓存的使用率。

4.2 合并表和分区表

分区表是 MySQL 5.1 的新特性, 而合并表已经有很长的历史了, 合并表和分区表的概念比较相似, 合并表是将许多个 MyISAM 表合并成一个续表, 类似于使用 `UNION` 语句将多个表合并, 合并表不是真的创建一个真正的表, 它就像一个用于放置相似表的容器。而分区表则通过一些特殊的语句, 创建独立的空间, 事实上创建分区表的每个分区都是有索引的独立表。分区看上去像一个单独的表, MySQL 在对分区表和合并表的实现上有很多共通之处。

4.2.1 合并表

MySQL 的合并表可以把多个结果相同的表合并成为一个续表, 事实上是容纳真正表的容器, 可以使用 `UNION` 语句来创建表, 下面是一个合并表的例子。

步骤 01 创建数据存储引擎是 MyISAM 类型的表 `mtable1` 和 `mtable2`, 命令如下:

```
mysql> create table mtable1(
->   data int not null primary key
-> )engine=myisam;
Query OK, 0 rows affected (0.03 sec)

mysql> create table mtable2(
->   data int not null primary key
-> )engine=myisam;
Query OK, 0 rows affected (0.02 sec)
```

步骤 02 向表 `mtable1` 和 `mtable2` 中插入数据, 命令如下:

```
mysql> insert into mtable1 values(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> insert into mtable2 values(2),(3),(4);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

步骤 03 使用 `UNION` 语句创建表 `mtable1` 和 `mtable2` 的合并表 `mergtable`, 命令如下:


```
mysql> create table mergtable(
  -> data int not null primary key
  -> )engine=merge union=(mtable1,mtable2) insert_method=last;
Query OK, 0 rows affected (0.03 sec)
```

insert_method=last 的含义是，如果向表 mergtable 中插入一条记录，那么就将这条记录插入到合并表所合并的最后一个表里面，以上例子就是将记录插入到 mtable2 表中。

步骤 04 查询合并表 mergtable 的信息，命令如下：

```
mysql> select *from mergtable;
+-----+
| data |
+-----+
| 1 |
| 2 |
| 2 |
| 3 |
| 3 |
| 4 |
+-----+
6 rows in set (0.00 sec)
```

值得注意的是合并表所包含的表列的数量和类型跟所合并的表的列的数量和类型都是一样的。同时也可以看到每个表的列有主键，这会导致合并表有重复的行，这是合并表的一个局限。

步骤 05 直接插入数据到 mergtable 中，命令如下：

```
mysql> insert into mergtable values(5);
Query OK, 1 row affected (0.00 sec)
```

步骤 06 查询 mtable1 表中的数据是否发生变化，命令如下：

```
mysql> select * from mtable1;
+-----+
| data |
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.00 sec)
```

从结果可以看出，数据没有发生变化。

步骤 07 查询 mtable2 表中的数据是否发生变化，命令如下：

```
mysql> select * from mtable2;
```

```

+-----+
| data |
+-----+
|    2 |
|    3 |
|    4 |
|    5 |
+-----+
4 rows in set (0.00 sec)

```

从结果可以看出，插入到合并表 `mergtable` 的一条数据记录已经插入到 `mtable2` 表中了。

步骤 08 删除表 `mtable1` 和 `mtable2`，命令如下：

```

mysql> drop table mtable1,mtable2;
Query OK, 0 rows affected (0.00 sec)

```

步骤 09 在 linux 环境查询 `mergtable`，发生错误，命令如下：

```

mysql> select * from mergtable;
ERROR 1146(42502): Table 'test.mtable1' doesn't exist;

```

值得注意的是，在 Windows 环境下的 MySQL 数据库做同样的操作，没有发生错误。MySQL 合并表的实现对性能有一定的影响，下面是一些需要注意的事项：

- 合并表看上去是一个表，事实上是逐个打开各个子表，这样的情况下，可能会因为缓存过多的表而导致超过 MySQL 缓存的最大设置。
- 创建合并表的 CREATE 语句不会检查子表是否兼容，如果创建了一个有效的合并表后对某个表进行了修改，那么合并表也会发生错误。

4.2.2 分区表

从 MySQL 5.1 版本开始支持数据表分区，通俗地讲，表分区是将一张大表，根据条件分割成若干小表。例如，某用户表的记录超过了 600 万条，那么就可以根据入库日期或者所在地将表分区。

笔者的数据库版本是 5.0.89，此时需要安装 MySQL 5.1 以上版本，这里以安装 `mysql-5.6.10-linux-i686-glibc23` 为例，具体操作步骤如下：

步骤 01 将 `mysql-5.6.10-linux-i686-glibc23.tat.gz` 解压。

```

[root@localhost tools]# gunzip mysql-5.6.10-linux-i686-glibc23.tar.gz
[root@localhost tools]# tar -xvf mysql-5.6.10-linux-i686-glibc23.tar

```

步骤 02 开始安装 MySQL 程序。

```

[root@localhost mysql-5.6.10-linux-i686-glibc23]# ./scripts/mysql_install_db
--user=root

```

步骤 03 修改/etc/mysql.cnf 中数据库配置选项，具体设置如下：

```
#####
# The MySQL server
[mysqld_multi]
mysqld      = /usr/local/mysql/bin/mysqld_safe
mysqladmin  = /usr/local/mysql/bin/mysqladmin
#user       = root
#password   = root

[mysqld1]
port        = 3306
socket      = /tmp/mysql.sock
pid-file    = /usr/local/var/mysql1/mysql1.pid
datadir     = /usr/local/var/mysql1
skip-locking
key_buffer  = 16M
max_allowed_packet = 1M
query_cache_type = on
query_cache_size = 999424
table_cache = 64
sort_buffer_size = 512K
net_buffer_length = 8K
read_buffer_size = 256K
read_rnd_buffer_size = 512K
myisam_sort_buffer_size = 8M
log-bin     = /usr/local/var/mysql1/mysql-bin.log
server-id   = 1
[mysqld2]
port        = 3307
socket      = /tmp/mysql2.sock
pid-file    = /usr/local/var/mysql2/mysql2.pid
datadir     = /usr/local/var/mysql2
skip-locking
key_buffer  = 16M
max_allowed_packet = 1M
table_cache = 64
sort_buffer_size = 512K
net_buffer_length = 8K
read_buffer_size = 256K
read_rnd_buffer_size = 512K
myisam_sort_buffer_size = 8M
server-id   = 2
#replicate-do-table=test.rep_t1
#replicate-ignore-table=test.rep_t2
#replicate-do-db=test
```



```

#replicate-do-db=cc
#replicate-ignore-db=tt
log-bin = /usr/local/var/mysql2/mysql-bin.log
[mysqld3]
port      = 3308
socket     = /tmp/mysql3.sock
pid-file= /usr/local/var/mysql3/mysql3.pid
datadir    = /usr/local/var/mysql3
skip-locking
key_buffer = 16M
max_allowed_packet = 1M
table_cache = 64
sort_buffer_size = 512K
net_buffer_length = 8K
read_buffer_size = 256K
read_rnd_buffer_size = 512K
myisam_sort_buffer_size = 8M
server-id = 3

#ssl
#ssl-ca=/etc/mysql/newcerts/ca-cert.pem
#ssl-cert=/etc/mysql/newcerts/server-cert.pem
#ssl-key=/etc/mysql/newcerts/server-key.pem
#####
[mysqld]
port      = 3309
socket     = /tmp/mysql.sock
skip-locking
key_buffer = 16M
max_allowed_packet = 1M
query_cache_size = 999424
table_cache = 64
sort_buffer_size = 512K
net_buffer_length = 8K
read_buffer_size = 256K
read_rnd_buffer_size = 512K
myisam_sort_buffer_size = 8M
server-id = 1

```

步骤 04 启动 MySQL，如下所示。

```

[root@localhost ~]# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.10-log MySQL Community Server (GPL)

```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql>
```

数据库分区是一种物理数据库设计技术，分区的主要目的是为了让某些特定的查询操作减少响应时间，同时对于应用来讲分区完全是透明的。MySQL 的分区主要有两种形式：水平分区（Horizontal Partitioning）和垂直分区（Vertical Partitioning）。

水平分区（Horizontal Partitioning）是根据表的行进行分割，这种形式的分区一定是通过表的某个属性作为分割的条件，例如，某张表里面数据日期为 2011 年的数据和日期为 2012 年的数据分割开，就可以采用这种分区形式。

垂直分区（Vertical Partitioning）是通过对表的垂直划分来减少目标表的宽度，是某些特定的列被划分到特定的分区。

通常可以通过下面命令查看是否支持分区，命令如下：

```
mysql> show variables like '%partition%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_partitioning | YES |
+-----+-----+
1 row in set (0.01 sec)
```

下面介绍 MySQL 各种分区表常用的操作案例。

1. RANGE 分区

RANGE 分区使用 `values less than` 操作符来进行定义，把连续且不相互重叠的字段分配给分区，命令如下。

```
mysql> create table emp(
-> empno varchar(20) not null,
-> empname varchar(20),
-> deptno int,
-> birthdate date,
-> salary int
-> )
-> partition by range(salary)
-> (
-> partition p1 values less than(1000),
-> partition p2 values less than(2000),
-> partition p3 values less than(3000)
-> );
Query OK, 0 rows affected (0.01 sec)

mysql> insert into emp values(1000,'kobe',12,'1888-08-08',1500);
```

```
Query OK, 1 row affected (0.01 sec)
```

```
mysql> insert into emp values(1000,'kobe',12,'1888-08-08',3500);
ERROR 1526 (HY000): Table has no partition for value 3500
```

此时，按照工资级别（字段 salary）进行表分区，partition by range 的语法类似于“switch...case”的语法，如果 salary 小于 1000，数据存储在 p1 分区；如果 salary 小于 2000，数据存储在 p2 分区；如果 salary 小于 3000，数据存储在 p3 分区。

上面插入的第二条数据工资级别（字段 salary）为 3500，此时没有分区用来存储该范围的数据，所以发生了错误。为了解决这种问题，加入“PARTITION p4 VALUES LESS THAN MAXVALUE”语句即可，命令如下。

```
mysql> drop table emp;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> create table emp(
-> empno varchar(20) not null,
-> empname varchar(20),
-> deptno int,
-> birthdate date,
-> salary int
-> )
-> partition by range(salary)
-> (
-> partition p1 values less than(1000),
-> partition p2 values less than(2000),
-> partition p3 values less than(3000),
-> partition p4 values less than maxvalue
-> );
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> insert into emp values(1000,'kobe',12,'1888-08-08',1000);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> insert into emp values(1000,'durant',12,'1888-08-08',3500);
Query OK, 1 row affected (0.00 sec)
```

maxvalue 表示最大的可能的整数值。值得注意的是 values less than 子句中也可以使用一个表达式，不过表达式结果不能为 NULL，下面按照日期进行分区，命令如下。

```
mysql> create table emp(
-> empno varchar(20) not null,
-> empname varchar(20),
-> deptno int,
-> birthdate date,
```



```

-> salary int
-> )
-> partition by range(year(birthdate))(
->   partition p0 values less than(1980),
->   partition p1 values less than(1990),
->   partition p2 values less than(2000),
->   partition p3 values less than maxvalue
-> );
Query OK, 0 rows affected (0.01 sec)

```

该方案中，生日 1980 年以前的员工信息存储在 p0 分区中，生日 1990 年以前的员工信息存储在 p1 分区中，生日 2000 年以前的员工信息存储在 p2 分区中，2000 年以后出生的员工信息存储在 p3 分区中。

RANGE 分区很有用，常常使用在以下几种情况。

(1) 如果要删除某个时间段的数据时，只需要删除分区即可。例如，要删除 1980 年以前出生员工的所有信息，此时会执行“alter table emp drop partition p0”的效率要比执行“delete from emp where year(birthdate)<=1980”高效得多。

(2) 如果使用包含日期或者时间的列可以考虑用到 RANGE 分区。

(3) 经常运行直接依赖于分割表的列的查询。比如，当执行某个查询，如“select count (*) from emp where year(birthdate) = 1999 group by empno”，此时 MySQL 数据库可以很迅速地确定只有分区 p2 需要扫描，这是因为查询条件对于其他分区不符合。

2. LIST 分区

LIST 分区类似 RANGE 分区，他们的区别主要在于，LIST 分区中每个分区的定义和选择是基于某列的值从属于一个集合，而 RANGE 分区是从属于一个连续区间值的集合。创建 LIST 分区命令如下：

```

mysql> create table employees(
->   empname varchar(20),
->   deptno int,
->   birthdate date not null,
->   salary int
-> )
-> partition by list(deptno)
-> (
->   partition p1 values in (10,20),
->   partition p2 values in (30),
->   partition p3 values in (40)
-> );
Query OK, 0 rows affected (0.01 sec)

```

以上示例以部门编号划分分区，10 号部门和 20 号部门的员工信息存储在 p1 分区，30 号

部门的员工信息存储在 p2 分区, 40 号部门的员工信息存储在 p3 分区, 同 RANGE 分区一样, 如果插入数据的部门编号不在分区值列表中时, 那么 “insert” 插入操作将失败并报错。

3. HASH 分区

HASH 分区是基于用户定义的表达式的返回值来进行选择的分区, 该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含 MySQL 中有效的、产生非负整数值的任何表达式。

HASH 分区主要用来确保数据在预先确定数目的分区中平均分布。在 RANGE 和 LIST 分区中, 必须明确指定一个给定的列值或列值集合应该保存在哪个分区中; 而在 HASH 分区中, MySQL 自动完成这些工作, 用户所要做的只是基于将要被哈希的列值指定一个列值或表达式, 以及指定被分区的表将要被分割成的分区数量。

先看下面的例子:

```
mysql> create table htable(
->   id int,
->   name varchar(20),
->   birthdate date not null,
->   salary int
-> )
-> partition by hash(year(birthdate))
-> partitions 4;
Query OK, 0 rows affected (0.00 sec)
```

当使用了 “PARTITION BY HASH” 时, MySQL 将基于用户函数结果的模数来确定使用哪个编号的分区。将要保存记录的分区编号为 $N = \text{MOD}(\text{表达式}, \text{num})$ 。如果表 htable 中插入一条 birthdate 为 “2010-09-23” 的记录, 可以通过如下方法计算该记录的分区。

```
mod(year('2010-09-23'), 4)
=mode(2010, 4)
=2
```

此时, 该条记录的数据将会存储在分区编号为 2 的分区空间。

4. 线性 HASH 分区

线性 HASH 分区和 HASH 分区的区别在于, 线性哈希功能使用的一个线性的 2 的幂运算法则, 而 HASH 分区使用的是哈希函数的模数。

先看下面的例子:

```
mysql> create table lhtable(
->   id int not null,
->   name varchar(20),
->   hired date not null default '1999-09-09',
->   deptno int
```

```

-> )
-> partition by linear hash(year(hired))
-> partitions 4;
Query OK, 0 rows affected (0.03 sec)

```

如果表 lhtable 中插入一条 hireddate 为 “2010-09-23” 的记录，记录将要保存的分区是 num 个分区中的分区 N，可以通过如下方法计算 N。

步骤 01 找到下一个大于 num 的 2 的幂，把这个值称作 V，可以通过下面的公式得到。 $V = \text{POWR}(2, \text{CEILING}(\text{LOG}(2, \text{num})))$ ，假设，num 的值是 13，那么 $\text{LOG}(2, 13)$ 就是 3.70043。 $\text{CEILING}(3.70043)$ 就是 4，则 $V = \text{POWER}(2, 4)$ ，即等于 16。

步骤 02 计算 $N = F(\text{column_list}) \& (V - 1)$ 此时当 $N \geq \text{num}$ 时， $V = \text{CEIL}(V/2)$ ，此时 $N = N \& (V - 1)$ 。

下面使用一个示例来说明通过线性哈希分区算法计算分区 N 的值，线性哈希分区表 t1 是通过下面的语句创建。

```

mysql> create table t1(
->   col1 int,
->   col2 char(5),
->   col3 date
-> )
-> partition by linear hash( year(col3) )
-> partitions 6;
Query OK, 0 rows affected (0.59 sec)

```

现在假设要插入两条记录到表 t1 中，其中一条记录 col3 列的值为 “2003-04-14”，另一条记录 cols 列值为 “1998-10-19”。第一条记录要保存到的分区计算过程如下：

记录将要保存到的分区 num 分区中的分区 N，假设 num 是 7 个分区，假设表 t1 使用线性 HASH 分区且有 4 个分区。

```

V = POWR(2, CEILING(LOG(2, num)))
V = POWR(2, CEILING(LOG(2, 7))) = 8
N = YEAR('2003-04-14') & (8 - 1)
  = 2003 & 7
  = 3

```

N 的值是 3，很显然 $3 \geq 4$ 不成立，所以附件条件不执行，所以第一条记录的信息将存储在 3 号分区中。

第二条记录将要保存到的分区序号计算如下：

```

V = POWR(2, CEILING(LOG(2, num)))
V = POWR(2, CEILING(LOG(2, 7))) = 8
= YEAR('1998-10-19') & (8 - 1)
= 1998 & 7

```


= 6

N 的值是 6，很显然 $6 \geq 4$ 成立，所以附件条件会执行。

```
V = CEIL(6/2) = 3
N = N & (V-1)
  = 6 & 2
  = 2
```

此时发现 $2 \geq 4$ 不成立，记录将被保存到#2 分区中。按照线性哈希分区的优点在于增加、删除、合并和拆分区将变得更加快捷，有利于处理含有极其大量（1000GB）数据的表。它的缺点在于，与使用常规 HASH 分区得到的数据分布相比，各个分区间数据的分布可能不大均衡。

5. KEY 分区

类似于 HASH 分区，区别在于 KEY 分区只支持计算一列或多列，且 MySQL 服务器提供其自身的哈希函数，这些函数是基于与 PASSWORD() 一样的运算法则。

先看下面的例子：

```
mysql> create table keytable(
-> id int,
-> name varchar(20) not null,
-> deptno int,
-> birthdate date not null,
-> salary int
-> )
-> partition by key(birthdate)
-> partitions 4;
Query OK, 0 rows affected (0.11 sec)
```

在 KEY 分区中使用关键字 LINEAR 和在 HASH 分区中使用具有同样的作用，分区的编号是通过 2 的幂算法得到，而不是通过模数算法。

6. 复合分区

复合分区是分区表中每个分区的再次分割，子分区既可以使用 HASH 分区，也可以使用 KEY 分区。这也被称为子分区。

复合分区需要注意以下问题：

- 如果一个分区中创建了复合分区，其他分区也要有复合分区。
- 如果创建了复合分区，每个分区中的复合分区数必有相同。
- 同一分区内的复合分区，名字不相同，不同分区内的复合分区名字可以相同。

下面通过案例讲述不同的复合分区的创建方法。

创建 RANGE - HASH 复合分区的命令如下：

```
mysql> create table rhtable
-> (
->   empno varchar(20) not null,
->   empname varchar(20),
->   deptno int,
->   birthdate date not null,
->   salary int
-> )
-> partition by range(salary)
-> subpartition by hash( year(birthdate) )
-> subpartition 3
-> (
->   partition p1 values less than (2000),
->   partition p2 values less than maxvalue
-> );
Query OK, 0 rows affected (0.23 sec)
```

创建 RANGE - KEY 复合分区的命令如下:

```
mysql> create table rktable(
->   no varchar(20) not null,
->   name varchar(20),
->   deptno int,
->   birth date not null,
->   salary int
-> )
-> partition by range(salary)
-> subpartition by key(birth)
-> subpartitions 3
-> (
->   partition p1 values less than (2000),
->   partition p2 values less than maxvalue
-> );
Query OK, 0 rows affected (0.07 sec)
```

创建 LIST - HASH 复合分区的命令如下:

```
mysql> create table lhtable(
->   no varchar(20) not null,
->   name varchar(20),
->   deptno int,
->   birth date not null,
->   salary int
-> )
-> partition by list(deptno)
-> subpartition by hash( year(birth) )
-> subpartitions 3
```

```

-> (
-> partition p1 values in (10),
-> partition p2 values in (20)
-> );
Query OK, 0 rows affected (0.08 sec)

```

创建 LIST-KEY 复合分区的命令如下：

```

mysql> create table lktable(
-> no varchar(20) not null,
-> name varchar(20),
-> deptno int,
-> birthdate date not null,
-> salary int
-> )
-> partition by list(deptno)
-> subpartition by key(birthdate)
-> subpartitions 3
-> (
-> partition p1 values in (10),
-> partition p2 values in (20)
-> );
Query OK, 0 rows affected (0.09 sec)

```

4.3 事务控制

MySQL 通过 SET AUTOCOMMIT、START TRANSACTION、COMMIT 和 ROLLBACK 等语句控制本地事务，具体语法如下。

```

START TRANSACTION | BEGIN [WORK];
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET AUTOCOMMIT = {0|1}

```

其中 START TRANSACTION 表示开启事务、COMMIT 表示提交事务、ROLLBACK 表示回滚事务、SET AUTOCOMMIT 用于设置是否自动提交事务。

默认情况下，MySQL 事务是自动提交的，如果需要通过明确的 COMMIT 和 ROLLBACK 再提交和回滚事务，那么需要通过明确的事务控制命令来开始事务，这是和 Oracle 的事务管理有明显不同的地方。如果应用从 Oracle 数据库迁移到 MySQL 数据库，则需要确保应用中是否对事务进行了明确的管理。

MySQL 的 AUTOCOMMIT（自动提交）默认是开启，对 MySQL 的性能有一定影响，举个例子来说，如果用户插入了 1000 条数据，MySQL 会提交事务 1000 次。这时可以把自动提

交关闭掉，通过程序来控制，只要一次提交事务就可以了。

可以通过如下方式关掉自动提交功能，命令如下：

```
mysql> set @@autocommit=0;
Query OK, 0 rows affected (0.00 sec)
```

查看自动提交功能是否被关闭，命令如下：

```
mysql> show variables like "autocommit";
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | ON    |
+-----+-----+
1 row in set (0.02 sec)
```

下面通过两个 Session (Session1 和 Session2) 来理解事务控制的过程，具体操作步骤如下：

步骤 01 在 Session1 中，打开自动提交事务功能，然后创建表 ctable 并插入两条记录。命令如下：

```
mysql> set @@autocommit=1;
Query OK, 0 rows affected (0.00 sec)

CREATE TABLE ctable (
data INT(4),
);

mysql> insert into ctable values(1);
Query OK, 1 row affected (0.02 sec)

mysql> insert into ctable values(2);
Query OK, 1 row affected (0.00 sec)
```

步骤 02 在 Session2 中，打开自动提交事务功能，然后查询表 ctable，命令如下：

```
mysql> set @@autocommit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from ctable;
+-----+
| data |
+-----+
| 1 |
| 2 |
+-----+
2 rows in set (0.00 sec)
```

步骤 03 在 Session1 中，关闭自动提交事务功能，然后向表 ctable 中插入两条记录，命令如下：

```
mysql> set @@autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into ctable values(3);
Query OK, 1 row affected (0.00 sec)

mysql> insert into ctable values(4);
Query OK, 1 row affected (0.00 sec)
```

步骤 04 在 Session2 中，查询表 ctable，命令如下：

```
mysql> select * from ctable;
+-----+
| data |
+-----+
| 1 |
| 2 |
+-----+
2 rows in set (0.00 sec)
```

从结果可以看出，在 session1 中新插入的两条记录没有查询出来。

步骤 05 在 Session1 中，提交事务，命令如下：

```
mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

步骤 06 在 Session2 中，查询表 ctable，命令如下：

```
mysql> select * from ctable;
+-----+
| data |
+-----+
| 1 |
| 2 |
| 3 |
| 4 |
+-----+
4 rows in set (0.00 sec)
```

如果在表的锁定期间，如果使用 START TRANSACTION 命令开启一个新的事务，会造成一个隐含的 unlock tables 被执行，该操作存在一定的隐患。下面通过一个案例来理解。

步骤 01 在 Session1 中，查询 nbaplayer 表，结果为空，命令如下：

```
mysql> select * from nbaplayer;
```

```
Empty set (0.00 sec)
```

步骤 02 在 Session2 中，查询 nbaplayer 表，结果为空，命令如下：

```
mysql> select * from nbaplayer;
Empty set (0.00 sec)
```

步骤 03 在 Session1 中，对表 nbaplayer 加写锁，命令如下：

```
mysql> lock table nbaplayer write;
Query OK, 0 rows affected (0.00 sec)
```

步骤 04 在 Session2 中，向表 nbaplayer 中增加一条记录，命令如下：

```
mysql> insert into nbaplayer values
(1, 'kobe', 10000);
```

步骤 05 在 Session1 中，插入一条记录，命令如下：

```
mysql> insert into nbaplayer values
(2, 'durant', 40000);
Query OK, 1 row affected (0.02 sec)
```

步骤 06 在 Session1 中，回滚刚才插入的记录，命令如下：

```
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)
```

步骤 07 在 Session1 中，开启一个新的事务，命令如下：

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

步骤 08 在 Session2 中，表锁被释放，此时成功增加该条记录，命令如下：

```
mysql> insert into nbaplayer values
(1, 'kobe', 10000);
Query OK, 1 row affected (2 min 32.99 sec)
```

步骤 09 在 Session2 中，查询 nbaplayer，命令如下：

```
mysql> select * from nbaplayer;
+-----+-----+-----+
| id   | name  | salary |
+-----+-----+-----+
| 2   | durant | 40000  |
| 1   | kobe  | 10000  |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

从结果可以看出，此时发现 Session1 的回滚操作并没有执行成功。

MySQL 提供的 LOCK IN SHARE MODE 锁可以保证会停止任何对它要读的数据行的更新

或者删除操作。下面通过一个例子来理解。

步骤 01 在 Session1 中, 开启一个新的事务, 然后查询数据表 nbaplayer 的 salary 列的最大值, 命令如下:

```
mysql> begin;
Query OK, 0 rows affected (0.00 sec)
mysql> select max(salary) from nbaplayer lock in share mode;
+-----+
| max(salary) |
+-----+
|          40000 |
+-----+
1 row in set (0.00 sec)
```

步骤 02 在 Session2 中, 尝试做更新操作, 命令如下:

```
mysql> update nbaplayer set salary = 90000 where id = 1;
等待
```

步骤 03 在 Session1 中, 提交事务, 命令如下:

```
mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

步骤 04 在 Session2 中, 等 Session1 的事务提交后, 此时更新操作成功执行, 结果如下:

```
mysql> update nbaplayer set salary = 90000 where id = 1;
Query OK, 1 row affected (16.25 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

4.4 MySQL 分布式事务

在 MySQL 中, 使用分布式事务的应用程序涉及一个或多个资源管理器和一个事务管理器, 分布式事务的事务参与者、资源管理器、事务管理等位于不同的节点上, 这些不同的节点相互协作共同完成一个具有逻辑完整性的事务。分布式事务的主要作用在于确保事务的一致性和完整性。

4.4.1 了解分布式事务的原理

资源管理器 (Resource Manager, 简称 RM) 用于向事务提供资源, 同时还具有管理事务提交或回滚的能力。数据库就是一种资源管理器。

事务管理器 (Transaction Manager, 简称 TM) 用于和每个资源管理器通信, 协调并完成

事务的处理。一个分布式事务中各个事务均是分布式事务的“分支事务”。分布式事务和各分支通过一种命名方法进行标识。

MySQL 执行分布式事务，首先要考虑网络中涉及多少个事务管理器，MySQL 分布式事务管理，简单地讲就是同时管理若干管理器事务的一个过程，每个资源管理器的事务当执行到被提交或者被回滚的时候，根据每个资源管理器报告的有关情况决定是否将这些事务作为一个原子性的操作执行全部提交或者全部回滚。因为 MySQL 分布式事务同时涉及多台 MySQL 服务器，所以在管理分布式事务的时候，必须要考虑网络可能存在的故障。

用于执行分布式事务的过程使用两个阶段。

(1) 第一阶段：所有的分支被预备。它们被事务管理器告知要准备提交，每个分支资源管理器记录分支的行动并指示任务的可行性。

(2) 第二阶段：事务管理器告知资源管理器是否要提交或者回滚。如果预备分支时，所有的分支指示它们将能够提交，那么所有的分支被告知提交。如果有一个分支出错，那么就全部都要回滚。特殊情况下，只有一个分支的时候，第二阶段则被省略。

分布式事务的主要作用在于确保事务的一致性和完整性。它利用分布式的计算机环境，将多个事务性的活动合并成一个事务单元，这些事务组合在一起构成原子操作，这些事务的活动要么一起执行并提交事务，要么回滚所有的操作，从而保证了多个活动之间的一致性和完整性。

4.4.2 分布式事务的语法

在 MySQL 中，执行分布式事务的语法格式如下：

```
XA {START|BEGIN} xid [JOIN|RESUME]
```

XA START xid 表示用于启动一个事务标识为 xid 的事务。xid 分布式事务表示的值既可以由客户端提供，也可以由 MySQL 服务器生成。

结束分布式事务的语法格式如下：

```
XA END xid [SUSPEND [FOR MIGRATE]]
```

其中 xid 包括：gtrid [, bqual [, formatID]], 含义如下：

- gtrid 是一个分布式事务标识符。
- bqual 表示一个分支限定符，默认值是空字符串。对于一个分布式事务中的每个分支事务，bqual 值必须是唯一的。
- formatID 是一个数字，用于标识由 gtrid 和 bqual 值使用的格式，默认值是 1。

```
XA PREPARE xid
```

该命令使事务进入 PREPARE 状态，也就是两个阶段提交的第一个阶段。

```
XA COMMIT xid [ONE PHASE]
```

该命令用来提交具体的分支事务。

XA ROLLBACK xid

该命令用来回滚具体的分支事务。也就是两阶段提交的第二个提交阶段，分支事务被实际地提交或者回滚。

XA RECOVER

该命令用于返回数据库中处于 PREPARE 状态的分支事务的详细信息。

分布式的关键在于如何确保分布式事务的完整性，以及在某个分支出现问题时如何解决故障。

分布式事务的相关命令就是提供给应用如何在多个独立的数据库之间进行分布式事务的管理，包括启动一个分支事务、使事务进入准备阶段以及事务的实际提交回滚操作等。

MySQL 分布式事务分为两类：内部分布式事务和外部分布式事务。内部分布式事务用于同一实例下跨多个数据引擎的事务，由二进制日志作为协调者；而外部分布式事务用于跨多个 MySQL 实例的分布式事务，需要应用层介入作为协调者，全局提交还是回滚，都是由应用层决定的，对应用层的要求比较高。

MySQL 分布式事务在某些特殊的情况下会存在一定的漏洞，当一个事务分支在 PREPARE 状态的时候失去了连接，在服务器重启以后，可以继续对分支事务进行提交或者回滚操作，没有写入二进制日志，这将导致事务部分丢失或者主从数据库不一致。

4.5 小结

本章主要讲解 MySQL 的一些高级特性，其中包括 MySQL 查询缓存，优化查询缓存来提高缓存命中率，并且详细介绍了 MySQL 合并表和分区，MySQL 提供的事务控制和锁定语法，并对 MySQL 事务管理和分布式事务进行了简单的介绍。值得注意的是 MySQL 分布式事务存在一定的漏洞，MySQL 分布式事务在特殊情况下是无法保证事务的完整性。

第 5 章

◀ MySQL 锁定机制 ▶

如何保证数据并发访问的一致性和有效性，是所有数据库必须解决的一个问题。另外，锁冲突也是影响数据库并发性能的一个重要的因素。应用程序在选择锁类型时，需要根据实际运行的需要，选择最佳的锁类型。

相对于其他数据库而言，MySQL 数据库的锁机制比较简单。不同的存储引擎，MySQL 数据库支持不同的锁机制，这也是 MySQL 数据库的一个显著特点。本章重点讨论 MySQL 锁机制的特点，以及解决 MySQL 锁问题的一些方法或建议。

5.1 MySQL 锁定机制概述

MySQL 与其他数据库在锁定机制方面最大的不同之处在于，对于不同的存储引擎支持不同的锁定机制。例如，InnoDB 存储引擎支持行级锁（row-level locking），也支持表级锁，默认的情况下是采用行级锁；MyISAM 和 MEMORY 存储引擎采用的是表级锁（table-level locking）。BDB 存储引擎采用的是页面锁（page-level-locking），同时也支持表级锁。

总的来说，MySQL 各存储引擎使用了三种级别的锁定机制：行级锁定、页级锁定和表级锁定，下面分析一下这三种级别的锁机制的特点。

1. 行级锁定

行级锁最大的特点是锁定对象的颗粒度很小，发生锁定资源争用的概率也很小，能够给予应用程序尽可能大的并发处理能力，从而提高一些需要高并发应用系统的整体性能。

虽然能够在并发处理能力上有较大的优势，但是行级锁也存在不少弊端。由于行级锁的颗粒度比较小，所以每次获取锁和释放锁会消耗比较大，因此加锁比较慢，很容易发生死锁。

行级锁定不是 MySQL 自己实现的锁定方式，而是由其他存储引擎所实现的，比如 InnoDB 存储引擎。InnoDB 实现了两种类型的行级锁，包括共享锁和排他锁，而在锁定机制的实现过程中为了让行级锁定和表级锁定共存，InnoDB 使用了两种内部使用的意向锁，也就是意向共享锁和意向排他锁。各个锁的含义如下：

- 共享锁（S）：允许一个事务读一行数据时，阻止其他的事务读取相同数据的排他锁。
- 排他锁（X）：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据的共享

锁和排他锁。

- 意向共享锁 (IS)：事务打算给数据行加行共享锁。事务在给一个数据行加共享锁前必须先取得该表的 IS 锁。
- 意向排他锁 (IX)：事务打算给数据行加行排它锁。事务在给一个数据行加排他锁前必须先取得该表的 IX 锁。

上面这 4 种锁的共存逻辑关系如表 5-1 所示。

表 5-1 4 种锁的共存逻辑关系表

| 锁模式 | 共享锁 (S) | 排他锁 (X) | 意向共享锁 (IS) | 意向排他锁 (IX) |
|------------|---------|---------|------------|------------|
| 共享锁 (S) | 兼容 | 冲突 | 兼容 | 冲突 |
| 排他锁 (X) | 冲突 | 冲突 | 冲突 | 冲突 |
| 意向共享锁 (IS) | 兼容 | 冲突 | 兼容 | 兼容 |
| 意向排他锁 (IX) | 冲突 | 冲突 | 兼容 | 兼容 |

如果一个事务请求的锁模式与当前的锁模式兼容，InnoDB 就将请求的锁授予该事务，如果两者不兼容，那么该事务就要等待锁释放。

意向锁是 InnoDB 存储引擎自动加的，对于普通 SELECT 语句，InnoDB 不会加任何锁，对于 INSERT、UPDATE、DELETE 语句，InnoDB 会自动给涉及的数据加排他锁，InnoDB 可以通过以下语句显示添加的共享锁和排他锁。

(1) 添加共享锁 (S) 的语句如下：

```
SELECT * FROM table_name WHERE ... LOCK IN SHARE MODE.
```

(2) 添加排他锁 (X) 的语句如下：

```
SELECT * FROM table_name WHERE ... FOR UPDATE.
```

共享锁和排他锁的详细用法本章后面的小节中会详细讲解。

2. 表级锁定

与行级锁不同的是，表级锁的锁定机制的颗粒度最大，该锁定机制的最大特点是系统开销比较小，由于实现逻辑非常简单，所以带来的系统的负面影响也最小。由于表级锁一次性将整个表锁定，因此可以很好地避免死锁的问题。

同时，表级锁定机制也存在一定的缺陷，由于表级锁的锁定机制颗粒很大，所以发生锁冲突的概率也最高，因此表级锁定机制下并发度也最低。

MySQL 数据库的表级锁定主要分为两种类型，一种是读锁定，另一种是写锁定，MySQL 数据库提供了四种队列来维护这两种锁，这 4 个队列间接地说明了数据库表级锁的四种状态，这 4 个队列如下。

- Current read lock queue (lock -> read)

- Padding read lock queue (lock -> read wait)
- Current write lock queue (lock -> write)
- Padding write lock queue (lock -> write wait)

其中 Current read lock queue 中存放的是当前持有读锁的所有线程，而正在等待资源的信息则存放在 Padding read lock queue 中。同样，Current write lock queue 中存放的是当前持有写锁的所有线程，而正在等待对资源写操作的信息则存放在 Padding write lock queue 中。

MySQL 内部实现读锁和写锁有多达 11 种具体的锁定类型，由系统中一个枚举类型变量 (thr_lock_type)定义，具体各种锁定类型如表 5-2 所示。

表 5-2 各种锁定类型的含义

| 锁定类型 | 含义 |
|-------------------------|----------------------------------------------------------------------------------------------|
| IGNORE | 当发生锁请求的时候内部交互使用，在锁定结构和队列中并不会有任何信息存储 |
| UNLOCK | 释放锁定请求的交互用锁类型 |
| READ | 普通读锁定 |
| WRITE | 普通写锁定 |
| READ_WITH_SHARED_LOCKS | 在 Innodb 中使用到，语法如下： SELECT ... LOCK IN SHARE MODE |
| READ_HIGH_PRIORITY | 高优先级读锁定 |
| READ_NO_INSERT | 不允许 Concurrent Insert 的锁定 |
| WRITE_ALLOW_WRITE | 这个类型实际上就是由存储引擎自行处理锁定的时候，mysql 允许其他的线程再获取读或者写锁定，因为即使资源冲突，存储引擎自行处理 |
| WRITE_ALLOW_READ | 这种锁定发生在对表 DDL 操作的时候，MySQL 可以允许其他线程获取读锁定，因为 MySQL 是通过重建整个表然后再 RENAME 而实现的该功能，所以整个过程表依然可以提供读服务 |
| WRITE_CONCURRENT_INSERT | 正在进行 Concurrent Insert 时锁使用的锁定方式，该锁定进行的时候，除了 READ_NO_INSERT 之外的其他任何读锁定请求都不会被阻塞 |
| WRITE_DELAYED | 在使用 INSERT DELAYED 时的锁定类型 |
| WRITE_LOW_PRIORITY | 显示声明的低级别锁定方式，通过设置 LOW_PRIORITY_UPDATE = 1 而产生 |
| WRITE_ONLY | 当在操作过程中某个锁定异常中断之后系统内部需要进行 CLOSE TABLE 操作，在这个过程中出现的锁定类型就是 WRITE_ONLY |

对于 MySQL 数据库读锁和写锁的加锁方式，通常使用 LOCK TABLE 和 UNLOCK TABLE 实现对表的加锁和解锁，表 5-3 是一个获得表锁和释放表锁的详细过程。

表 5-3 一个获得表锁和释放表锁的例子

| Session1 | Session2 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 获得表 lock_table_test 的 READ 锁，命令如下： mysql> lock table lock_table_test read; Query OK, 0 rows affected (0.00 sec) | |
| 当前 Session1 可以查询出该表的记录，命令如下： mysql> select *from lock_table_test limit 0,1; +-----+-----+ id data +-----+-----+ 1 t +-----+-----+ 1 row in set (0.00 sec) | Session2 也可以查询出该表的记录，命令如下： mysql> select *from lock_table_test limit 0,1; +-----+-----+ id data +-----+-----+ 1 t +-----+-----+ 1 row in set (0.00 sec) |
| | Session2 更新锁定表将会发生等待获得锁，命令如下： mysql> update lock_table_test set data='test' where id = 1; 等待 |
| 释放锁，命令如下： mysql> unlock tables; Query OK, 0 rows affected (0.00 sec) | 等待 |
| | Session2 获得锁，更新操作执行完成，命令如下： mysql> update lock_table_test set data='test' where id = 1; Query OK, 1 rows affected (1 min 1.77 sec) Rows matched: 1 Changed: 1 Warnings: 0 |

3. 页级锁定

页级锁定在 MySQL 中是比较特殊的一种锁定机制，页级锁定的特点是颗粒度介于行级锁定与表级锁定之间，所以获取锁定所需要的资源开销，以及锁提供的并发处理的能力也介于表级锁定和行级锁定之间。

在数据库实现资源锁定的过程中，锁定机制的粒度越小，数据库实现的算法越复杂，数据库所消耗的内存也越大。不过，随着锁机制粒度越来越小，应用的并发发生锁等待的几率也越来越小，系统整体性能也随之增高。

MySQL 是用写队列和读队列来完成对数据库的读和写的操作的，所以说，MySQL 数据库存在读锁和写锁的概念，对于写锁而言，如果表没有加锁，那么对其表加写锁。如果表已经加了写锁，此时会将写操作的请求放入写锁的队列中。而对于读锁而言，如果没有加入读锁，那么请求会加入一个读操作的锁，其他读操作的请求会放到读锁的队列中。

下面通过一个简单的例子来说明读写操作，具体操作步骤如下：

步骤 01 首先创建表 content，语句如下：

```
mysql> use test;
Database changed
mysql> create table content(id int,content varchar(20));
Query OK, 0 rows affected (0.53 sec)
```

步骤 02 向表 content 里面添加多条数据，数据越多，效果越明显，语句如下：

```
mysql> insert content select * from content;
Query OK, 262144 rows affected (4.83 sec)
Records: 262144 Duplicates: 0 Warnings: 0

mysql> insert content select * from content;
Query OK, 524288 rows affected (8.58 sec)
Records: 524288 Duplicates: 0 Warnings: 0

mysql> insert content select * from content;
Query OK, 1048576 rows affected (18.31 sec)
Records: 1048576 Duplicates: 0 Warnings: 0
```

步骤 03 此时，准备工作已经完成，根据下表运行读写操作，理解 MySQL 读写队列运行的过程，如表 5-4 所示。

表 5-4 运行 MySQL 读写队列的过程

| Session1 | Session2 |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| mysql>select count(*) from content; | |
| | 此时执行 mysql> update content set content = 'test2' where id = 1; |
| | 等待 |
| 查询结束 | |
| | mysql> update content set content = 'test2' where id = 1; Query OK, 0 rows affected (5.16 sec) Rows matched: 1048576 Changed: 0 Warnings: 0 |
| Session1 关闭 | |
| | mysql> update content set content = 'test2' where id = 1; Query OK, 0 rows affected (3.14 sec) Rows matched: 1048576 Changed: 0 Warnings: 0 |

对于 session1，此时表没有加锁，那么此时 session1 操作对表加上一个读锁，对于 session2 此时表 content 因为已经加上了读锁，所以会将 update 请求放到锁定队列中。

从上述特点可见，很难笼统地说哪种锁定机制好，只能根据具体应用的特点来选择哪种锁定机制更合适。仅从锁的角度来看，表级锁更适合以查询为主，只有少量按索引条件更新数据的应用。而行级锁适用于有大量按索引条件并发更新少量不同数据，同时又有并发查询的应用。

5.2 MyISAM 表级锁

MySQL 的 MyISAM 存储引擎只支持表级锁，随着应用对事务的完整性和并发性要求越来越高，MySQL 才出现基于支持事务的存储引擎，后来慢慢出现了支持行级锁的 InnoDB 存储引擎和支持页级锁的 BDB 存储引擎。

MyISAM 存储引擎基本上可以说是对 MySQL 所提供的锁定机制所实现的表级锁定依赖最大的一种存储引擎了。而其他几种支持事务的存储引擎，如 InnoDB、BDB、NDB Cluster 存储引擎则是让 MySQL 将锁定的处理直接交给存储引擎自己来处理。

目前 MyISAM 的表锁是使用最为广泛的锁类型，本节详细介绍 MyISAM 表锁的使用。

5.2.1 MyISAM 表级锁的锁模式

MySQL 的表级锁有两种模式：表共享读锁（table read lock）和表独占写锁（table write lock）。

锁模式的兼容性如表 5-5 所示。

表 5-5 锁模式的兼容性

| | None | 读 锁 | 写 锁 |
|----|------|-----|-----|
| 读锁 | 兼容 | 兼容 | 冲突 |
| 写锁 | 兼容 | 冲突 | 冲突 |

对于 MyISAM 表的读操作不会因为不同进程访问资源而发生阻塞，而对于 MyISAM 表的写操作会阻塞其他用户对同一表的读和写操作。

下面通过一个例子来看下 MySQL 表锁的读锁的过程，如表 5-6 所示。

表 5-6 MySQL 表锁的读锁的过程

| Session1 | Session2 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| <p>首先新建表 read_lock，然后添加一条数据，命令如下：</p> <pre>mysql> create table read_lock(-> id int, -> data varchar(20) ->)engine=myisam default charset=utf8; Query OK, 0 rows affected (0.01 sec) mysql>insert into read_lock values(102,'data'); Query OK, 1 row affected (0.01 sec)</pre> | |

(续表)

| Session1 | Session2 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>获得 readk lock 表的读锁，命令如下：</p> <pre>mysql> lock table read_lock read; Query OK, 0 rows affected (0.00 sec)</pre> <p>查询 read_lock 表的数据，命令如下：</p> <pre>mysql> select *from read_lock; +-----+-----+ id data +-----+-----+ 102 data +-----+-----+ 1 row in set (0.00 sec)</pre> <p>执行更新操作，发现不能够执行，命令如下：</p> <pre>mysql> update read_lock set data='test' where id = 102; ERROR 1099 (HY000): Table 'read_lock' was locked with a READ lock and can't be updated</pre> <p>插入数据时不能执行，执行命令如下：</p> <pre>mysql>insert into read_lock values(13,'data2'); ERROR 1099 (HY000): Table 'read_lock' was locked with a READ lock and can't be updated<p>释放读锁，命令如下：</p><pre>mysql> unlock tables; Query OK, 0 rows affected (0.00 sec)</pre></pre> | <p>其他 session 是可以查询数据的，如下：</p> <pre>mysql> select *from read_lock; +-----+-----+ id data +-----+-----+ 102 data +-----+-----+ 1 row in set (0.00 sec)</pre> <pre>mysql> insert into read_lock values(13,'data3'); 等待</pre> |
| | <p>成功添加一条数据，命令如下：</p> <pre>mysql> insert into read_lock values(13,'data3'); Query OK,1 row affected(3 min 26.30 sec)</pre> |

下面看下 MySQL 表级锁的写锁测试的例子，如表 5-7 所示。

表 5-7 一个 MySQL 表锁的写锁测试的例子

| Session1 | Session2 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| <p>获得表 read_lock 表的写锁，命令如下：</p> <pre>mysql> lock table read_lock write; Query OK, 0 rows affected (0.00 sec)</pre> <p>当前 session 对锁定表的查询、更新、插入操作都是可以执行的，执行命令如下：</p> <pre>mysql> select * from read_lock; +-----+-----+ id data +-----+-----+ 102 data 103 data3 +-----+-----+ 2 rows in set (0.00 sec)</pre> <pre>mysql> update read_lock set data='tt' where id = 103; Query OK, 1 row affected (0.05 sec) Rows matched: 1 Changed: 1 Warnings: 0</pre> <pre>mysql> insert into read_lock values(104,'data4'); Query OK, 1 row affected (0.00 sec)</pre> | <p>Session2 对锁定表的查询被阻塞，需要等待锁释放，执行命令如下：</p> <pre>mysql> select * from read_lock; 等待</pre> |

(续表)

| Session1 | Session2 |
|----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 释放锁，命令如下： mysql> unlock tables; Query OK, 0 rows affected (0.00 sec) | |
| | Session2 获得锁，查询返回结果，命令如下： mysql> select * from read_lock; +-----+-----+ id data +-----+-----+ 102 data 103 tt 104 data4 +-----+-----+ 3 rows in set (54.45 sec) |

5.2.2 获取 MyISAM 表级锁的争用情况

MyISAM 存储引擎只支持表锁，MySQL 数据库可以通过检查 table_locks_waited 和 table_locks_immediate 状态变量来分析系统上的表锁的争夺情况，命令如下：

```
mysql> show status like 'table%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 120 |
| Table_locks_waited | 0 |
+-----+-----+
2 rows in set (0.25 sec)
```

这里有两个状态变量记录了 MySQL 内部表级锁定的情况，两个变量的含义如下：

- Table_locks_immediate: 产生表级锁定的次数。
- Table_locks_waited: 出现比较锁定争用而发生等待的次数。

如果 Table_locks_waited 的值比较高，那么说明了存在着比较严重的表级锁争用情况。MyISAM 在读操作占主导的情况下是高效的，可一旦出现大量读写操作并发，同 InnoDB 相比，MyISAM 的执行效率就会直线下降。对于 MyISAM 存储引擎表，新的数据会被附加到数据文件的结尾，可如果经常做一些 UPDATE 和 DELETE 操作，数据将不会是连续的，数据文件中就会出现很多空洞，此时再插入新的数据时，默认情况下会先看这些空洞是否可以容纳新的数据，如果可以容纳新的数据，那么会将数据保存到空洞里面去，反之，会将新的数据保存到数据文件的结尾。这样做是为了减少文件的大小，降低产生文件碎片的可能。

MyISAM 存储引擎表往往因为读表请求的增加，会出现比较严重的读写锁的问题，所以

经常在实际应用中采用主从分离，主从服务器读写操作分离出来，主服务器执行写操作，而从服务器负责查询操作，此时往往因为主服务器执行完了写入的操作，但从服务器有大量的查询操作，会被这些来自主辅同步的 UPDATE 和 INSERT 操作严重堵塞，最后造成所有的 MySQL 从库负载迅速上升。在解决 MyISAM 读写互斥问题中，由于没有办法在短期内增加读的服务器，所以通过对 MySQL 进行一些配置，以牺牲数据实时性为代价，来换取所有服务器的安全。具体配置如下。

(1) 当对于同一个 MyISAM 表进行查询和插入操作时，为了降低锁竞争频率，可以将 `concurrent_insert` 的值设置为 2，此时不管表有没有空洞，都允许数据文件结尾并发插入数据，至于产生的文件碎片，可以定期使用 `OPTIMIZE TABLE` 语法进行优化。

(2) 在默认情况下，写操作的优先级要高于读操作的优先级别，即便是先发送的是读操作请求，后发送是写操作的请求，此时也会优先处理写请求，然后处理读请求。可以考虑设置 `max_write_lock_count=1`，此时当系统处理一个写操作后，就会暂停写操作，给读操作执行的机会。

(3) 降低写操作的优先级，给读操作更高的优先级别，可以将 `low-priority-updates` 设置为 1。

5.2.3 MyISAM 表级锁加锁方法

MyISAM 在执行查询语句的时候，会自动给查询语句涉及的数据库表记录添加读锁，而在执行数据更新操作，比如执行 UPDATE、INSERT、DELETE 等语句的时候，MySQL 数据库会自动给涉及的表记录添加写锁。

对于 MySQL 数据库读锁和写锁的加锁方式，通常使用命令 `LOCK TABLE` 和 `UNLOCK TABLE` 实现手动加表级锁，`LOCK TABLE`、`UNLOCK TABLE` 可以用来锁定和释放当前线程的表，具体语法如下。

```
LOCK TABLE
tbl_name [As alias] {READ[LOCAL] | [LOW_PRIORITY] WRITE}
[, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
UNLOCK TABLES
```

要想使用 `LOCK TABLE`，必须拥有有关表的 `LOCK TABLE` 权限和 `SELECT` 权限，如果一个线程获得对一个表的读锁，该线程只能从该表中读取数据；如果一个线程获得对一个表的写锁，那么该线程只可以对表进行写数据，此时其他线程会被阻塞，直到写被释放为止。

下面详细讲解下 `LOCK TABLE` 的用法。

当对表以别名的形式锁定时，不能在一次查询中多次使用一个已锁定的表使用别名代替，在此情况下，必须分别获得对每个别名的锁定，举例如下：

```
mysql> lock table c_table write, c_table as c write;
Query OK, 0 rows affected (0.22 sec)
```



```
mysql> insert into c_table select * from c_table;
ERROR 1100 (HY000): Table 'c_table' was not locked with LOCK TABLES

mysql> insert into c_table select *from c_table as c;
Query OK, 4 rows affected (0.06 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

如果查询一个别名引用一个表，那么必须使用相同的别名锁定该表，如果没有别名，则不会锁定该表，举例如下：

```
mysql> lock table c_table read;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from c_table as c;
ERROR 1100 (HY000): Table 'c' was not locked with LOCK TABLES
```

如果使用一个别名锁定一个表，那么必须使用该别名在查询中引用该表。举例如下：

```
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)

mysql> lock table c_table as c read;
Query OK, 0 rows affected (0.00 sec)

mysql> select *from c_table;
ERROR 1100 (HY000): Table 'c_table' was not locked with LOCK TABLES

mysql> select *from c_table as c;
+-----+-----+
| id   | data |
+-----+-----+
| 1   | test |
| 2   | data2 |
| 3   | data3 |
| 4   | data4 |
| 1   | test |
| 2   | data2 |
| 3   | data3 |
| 4   | data4 |
+-----+-----+
8 rows in set (0.01 sec)
```

READ LOCAL 和 READ 之间的区别是，READ LOCAL 允许在锁定被保持时，执行非冲突性 INSERT 语句同时插入，其作用就是在满足 MyISAM 表并发插入条件的情况下，允许其他用户在表尾并发插入记录。

如果对一个表使用 `LOW_PRIORITY WRITE` 锁定，这意味着，MySQL 等待特定的锁定，直到没有申请 `READ` 锁定的线程时为止。当线程已经获得 `WRITE` 锁定，并在等待得到锁定表清单中的用于下一个表的锁定时，所有其他线程会等待 `WRITE` 锁定被释放。

5.2.4 MyISAM Concurrent Insert 的特性

MyISAM 存储引擎有个系统变量 `concurrent_insert`，专门用以控制其并发插入的行为，其值分别是 0、1、2。

- 当 `concurrent_insert` 的值为 0 时候，不允许并发插入。
- 当 `concurrent_insert` 的值为 1 时候，如果表中没有被删除的行，MyISAM 允许在一个进程读表的同时，另一个进程从表尾插入记录。
- 当 `concurrent_insert` 的值为 2 时候，无论表中有没有被删除的行，都允许在表尾并发插入记录。

MySQL 数据库默认的 `concurrent_insert` 的值为 1，即 MySQL 允许一个进程读表的同时，另一个进程从表尾插入记录。查询 `concurrent_insert` 值的命令如下：

```
mysql> show variables like '%concurrent%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| concurrent_insert | 1 |
+-----+-----+
1 row in set (0.08 sec)
```

在如表 5-8 所示的例子中，Session1 获得表 `c_table` 的 `READ LOCAL` 锁，此时 Session2 可以插入数据记录，但是更新操作发生阻塞，此时，Session1 不能对表进行插入和更新数据操作。等到 Session1 释放锁资源后，Session2 完成更新操作。

表 5-8 MyISAM 存储引擎并发操作的例子

| Session1 | Session2 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| <div>新建表 <code>c_table</code>，命令如下： <pre>mysql> use test; Database changed mysql> create table c_table(-> id int, -> data varchar(20) ->)engine=myisam default charset=utf8; Query OK, 0 rows affected (0.09 sec) mysql> insert into c_table values(1,'data'); Query OK, 1 row affected (0.08 sec)</pre></div> | |

(续表)

| Session1 | Session2 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>获得表 c_table 的 READ LOCAL 锁, 命令如下:</p> <pre>mysql> lock table c_table read local; Query OK, 0 rows affected (0.02 sec)</pre> <p>当前 session 不能对表进行插入数据操作, 执行结果如下:</p> <pre>mysql> insert into c_table values(3,'data3'); ERROR 1099 (HY000): Table 'c_table' was locked with a READ lock and can't be updated</pre> <p>当前 session 不能对表进行更新数据操作, 执行结果如下:</p> <pre>mysql> update c_table set data='test' where id = 1; ERROR 1099 (HY000): Table 'c_table' was locked with a READ lock and can't be updated</pre> | <p>Session2 可以进行插入操作, 但是更新会等待, 执行结果如下:</p> <pre>mysql> insert into c_table values(3,'data3'); Query OK, 1 row affected (0.00 sec)</pre> <pre>mysql> update c_table set data='test' where id = 1; 等待</pre> |
| <p>Session1 查询不能访问其他 session 插入的数据, 执行结果如下:</p> <pre>mysql> select * from c_table; +-----+-----+ id data +-----+-----+ 1 test 2 data2 +-----+-----+ 2 rows in set (0.00 sec)</pre> | |
| <p>释放锁, 执行结果如下:</p> <pre>mysql> unlock tables; Query OK, 0 rows affected (0.00 sec)</pre> <p>当前 session 解锁后可以查询到其他 session 插入的记录, 执行结果如下:</p> <pre>mysql> select * from c_table; +-----+-----+ id data +-----+-----+ 1 test 2 data2 3 data3 +-----+-----+ 3 rows in set (0.00 sec)</pre> | <p>Session2 获得锁, 更新操作完成, 执行结果如下:</p> <pre>mysql> update c_table set data='test' where id = 1; Query OK, 1 row affected (2 min 3.42 sec) Rows matched: 1 Changed: 1 Warnings: 0</pre> |

5.2.5 MyISAM 表锁优化建议

对于 MyISAM 存储引擎，虽然使用表级锁在实现过程比行级锁和页级锁带来的附加成本要小，所消耗的资源也是最小的，不过 MyISAM 表级锁的颗粒比较大，在数据库并发处理过程中产生的数据资源争用的情况也会比其他的锁定级别都要多，从而在较大程度上会降低并发处理能力。MyISAM 表锁的优化建议如下：

(1) MyISAM 表锁的锁定级别是固定的，所以在考虑 MyISAM 表级锁优化时，重点考虑如何提高并发的效率。

(2) 减少锁定的时间，尽量让查询的时间尽可能的短。减少比较复杂的查询语句，可以考虑将复杂的查询分解成多个小的查询。尽可能建立足够高效的索引，让数据检索更迅速。尽量让 MyISAM 存储引擎的表控制字段类型。利用合理的机会优化 MyISAM 表数据文件。

(3) MyISAM 表级锁可以考虑分离能并行的操作，对于读锁互相阻塞的表级锁，可能会觉得在存储引擎的表上就只能是完全的串行化，没有办法再并行了，可是 MyISAM 的存储引擎还有个非常有用的特性，就是 Concurrent Insert 的特性。可以考虑设置 Concurrent Insert 值为 2，此时无论 MyISAM 存储引擎的数据文件的中间部分是否存在空洞（因为删除数据而留下的空闲空间），都允许数据文件尾部进行插入操作。

(4) MyISAM 的表级锁定对于读和写是有不同优先级别设定的，默认情况下写操作的优先级别高于读操作的优先级别，可以考虑根据应用的实际情况来设置读锁和写锁的优先级别，此时可以通过设置系统参数 `low_priority_updates=1`，将写的优先级别设置比读的优先级低。

5.3 InnoDB 行级锁

MySQL 数据库最常见的两种存储引擎是 MyISAM 和 InnoDB 两种存储引擎，这两种类型的存储引擎的表各有优缺点，MyISAM 类型不支持事务处理，不过执行的效率比 InnoDB 类型的存储引擎更快，而 InnoDB 类型的存储引擎支持事务特性，并且 InnoDB 提供外键等数据库高级功能。在处理数据量上 InnoDB 可以处理海量数据，并且在具有良好索引的基础上，InnoDB 的查询速度要比 MyISAM 要快。另外，InnoDB 存储引擎采用了行级锁，本节会详细介绍 InnoDB 存储引擎的行级锁。

5.3.1 InnoDB 行级锁模式

InnoDB 存储引擎支持行级锁，支持事务处理。事务是由一组 SQL 语句组成的逻辑处理单元，它的 ACID 特性如下：

- 原子性 (Atomicity)：事务具有原子不可分割的特性，要么一起执行，要么都不执行。
- 一致性 (Consistency)：在事务开始和事务结束时，数据都保持一致状态。这意味着

多相关的数据规则都必须应用于事务的修改,以保证事务的完整性。

- 隔离性 (Isolation): 在事务开始和结束过程中,事务保持着一定的隔离特性,保证事务在不受外部并发数据操作的影响。
- 持久性 (Durability): 事务完成后,数据将会被持久化到数据库中。

InnoDB 存储引擎并发事务处理能力大大增加了数据库资源的利用率,提高了数据库系统的事务吞吐量,但并发事务同时也存在一些问题,主要包括:更新丢失(Lost Update)、脏读(Dirty Reads)、不可重复读(Non-Repeatable Reads)、幻读(Phantom Reads)。它们的具体含义如下:

- 更新丢失 (Lost Update): 两个事务更新同一行数据,但是第二个事务却中途失败退出了,导致对两个修改都失效了,这时系统没有执行任何锁操作,因此并发事务并没有被隔离。
- 脏读 (Dirty Reads): 一个事务读了某行数据,但是另一个事务已经更新了这行数据,这是非常危险的,很可能所有的操作被回滚。
- 不可重复读 (Non-Repeatable Reads): 一个事务对一行数据重复读取两次,可是得到了不同的结果。在两次读取数据的中途,有可能存在另外一个事务对数据进行了修改。
- 幻读 (Phantom Reads): 事务在操作过程中进行两次查询,第二次查询结果包含了第一次没有出现的数据,出现幻读的主要原因是,两次查询过程中另一个事务插入新的数据。

数据库并发中的“更新丢失”通常是应该完全避免的。但防止更新丢失数据,并不能单靠数据库事务控制来解决,需要应用程序对要更新的数据加必要的锁来解决,而以上出现的数据库“脏读”、“不可重复读”、“幻读”都必须由数据库提供一定的事务隔离机制来解决。为了避免数据库事务带来的问题,在标准 SQL 规范中定义了 4 个事务的隔离级别,不同的隔离级别对事务处理不一样。

数据库隔离级别包括:未提交读 (Read uncommitted)、已提交读 (Read committed)、可重复读 (Repeatable read)、可序列化 (Serializable),事务的隔离级别越严格,并发副作用就越小,但付出的代价也越大,这 4 种数据库隔离级别的比较如表 5-9 所示。

表 5-9 数据库隔离级别的比较

| 读数据一致性及允许的 并发副作用 隔离级别 | 读数据一致性 | 脏读 | 不可重复读 | 幻读 |
|-----------------------------|----------------------|----|-------|----|
| 未提交读 (Read uncommitted) | 最低级别,只能保证不读取物理上损坏的数据 | 是 | 是 | 是 |
| 已提交读 (Read committed) | 语句级 | 否 | 是 | 是 |
| 可重复读 (Repeatable read) | 事务级 | 否 | 否 | 是 |
| 可序列化 (Serializable) | 最高级别,事务级 | 否 | 否 | 否 |

InnoDB 存储引擎实现了四种行锁，分别是：共享锁(S)、排它锁(X)、意向共享锁(IS)、意向排他锁(IX)，下面进一步学习这四种行锁的知识。

首先，使用共享锁和排他锁必须要满足以下几个条件。

- 设置 autocommit 的值是 OFF 或者 0。
- 表的数据引擎是支持事务的，比如 InnoDB 数据引擎。
- 如果不管 autocommit，手动在事务里执行操作，这个时候要使用 begin 或者 start transaction 开始事务。
- 不要在锁定事务规定的时间外使用共享锁和排他锁。

下面通过一个例子来理解。首先建立一个表，然后添加记录，命令如下：

```
mysql> create table s_table(  
-> id int,  
-> data varchar(20)  
-> )engine=innodb;  
Query OK, 0 rows affected (0.13 sec)  
  
mysql> insert into s_table values(120,'test');  
Query OK, 1 row affected (0.00 sec)
```

使用 InnoDB 存储引擎共享锁的过程如表 5-10 所示。

表 5-10 一个 InnoDB 存储引擎共享锁的例子

| Session1 | Session2 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 关闭自动事务提交，查看数据表 s_table 中 id 为 120 的数据，命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> use test; Database changed mysql> select id,data from s_table where id=120; +-----+-----+ id data +-----+-----+ 120 tt +-----+-----+ 1 row in set (0.00 sec) | 关闭自动事务提交，查看数据表 s_table 中 id 为 120 的数据，命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> use test; Database changed mysql> select id,data from s_table where id = 120; +-----+-----+ id data +-----+-----+ 120 tt +-----+-----+ 1 row in set (0.00 sec) |
| 当前 session 对 id=120 的这条记录添加共享锁 命令如下： mysql> select id,data from s_table where id = 120 lock in share mode; +-----+-----+ id data +-----+-----+ 120 tt +-----+-----+ 1 row in set (0.00 sec) | |

(续表)

| Session1 | Session2 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | Session2 可以查询到 id=120 的该条记录，同时也可以对该条记录加共享锁，命令如下： mysql> select id,data from s_table where id = 120 lock in share mode; +-----+-----+ id data +-----+-----+ 120 tt +-----+-----+ 1 row in set (0.00 sec) |
| 当前 session 对该条记录进行更新操作，此时会发生等待，命令如下： mysql> update s_table set data='test2' where id = 120; 等待 | |
| | 当前 session 同样使用 update 语句进行更新操作，则会导致死锁退出，命令如下： mysql> update s_table set data='test2' where id = 120; ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction |
| 获得锁之后，可以更新成功，命令如下： mysql> update s_table set data='test2' where id = 120; Query OK, 1 row affected (10.38 sec) Rows matched: 1 Changed: 1 Warnings: 0 | |

下面接着学习 InnoDB 存储引擎使用排他锁的例子，具体过程如表 5-11 所示。

表 5-11 一个 InnoDB 存储引擎排他锁的例子

| Session1 | Session2 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 关闭自动事务提交 查看数据表 s_table 中 id 为 120 的数据，命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> use test; Database changed mysql> select id,data from s_table where id = 120; +-----+-----+ id data +-----+-----+ 120 tt +-----+-----+ 1 row in set (0.00 sec) | 关闭自动事务提交 查看数据表 s_table 中 id 为 120 的数据 命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> use test; Database changed mysql> select id,data from s_table where id = 120; +-----+-----+ id data +-----+-----+ 120 tt +-----+-----+ 1 row in set (0.00 sec) |

(续表)

| | |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div>Session1</div> <div>当前session对id=120 的这条记录加for update 的排他锁, 命令如下: mysql> select id,data from s_table where id = 120 for update; +-----+-----+ id data +-----+-----+ 120 tt +-----+-----+ 1 row in set (0.00 sec)</div> | <div>Session2</div> <div>当前 session 直接查询 select id,data from s_table where id = 120 该条记录, 命令如下: mysql> select id,data from s_table where id = 120; +-----+-----+ id data +-----+-----+ 120 test2 +-----+-----+ 1 row in set (0.00 sec)</div> <div>如果当前 session 对 id=120 的这条记录加 for update 的排他锁会发生等待, 命令如下: mysql> select id,data from s_table where id = 120 for update; 等待</div> |
| <div>接下来当前 session 更新该条记录, 然后提交事务, 事务提交后会释放锁, 命令如下: mysql> update s_table set data = 'test4' where id = 120; Query OK, 1 row affected (0.00 sec) Rows matched: 1 Changed: 1 Warnings: 0 mysql> commit; Query OK, 0 rows affected (0.01 sec)</div> | <div>当前 session2 得到 session1 提交的记录 命令 如下: mysql> select id,data from s_table where id = 120 for update; +-----+-----+ id data +-----+-----+ 120 test4 +-----+-----+ 1 row in set (12.11 sec)</div> |

5.3.2 获取 InnoDB 行级锁的争用情况

对于 InnoDB 所使用的行级锁定，系统中是通过另外一组更为详细的状态来记录的，查看命令如下：

```
mysql> show status like '%innodb_row_lock%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Innodb_row_lock_current_waits | 0      |
| Innodb_row_lock_time      | 211656 |
| Innodb_row_lock_time_avg  | 26457  |
| Innodb_row_lock_time_max  | 50968  |
| Innodb_row_lock_waits     | 8      |
+-----+-----+
5 rows in set (0.00 sec)
```

InnoDB 的行级锁定状态不仅记录了锁定等待次数，还记录了锁定总时间长，每次平均时长，以及最大的时长，各个状态变量的说明如下：

- `Innodb_row_lock_current_waits`：当前正在等待锁定的数量。
- `Innodb_row_lock_time`：从系统启动到现在锁定总时间长度。
- `Innodb_row_lock_time_avg`：每次等待锁花的平均时间。
- `Innodb_row_lock_time_max`：从系统启动到现在等待最常的一次所花的时间。
- `Innodb_row_lock_waits`：系统启动后到现在总共等待的次数。

根据 InnoDB 提供的这些系统状态的分析，制订相应的优化计划，尤其是当等待次数比较高的时候，而且每次等待时间也比较大的时候，就需要分析系统出现这种情况的原因。

此外，还可以通过如下方法来监控 InnoDB 行级锁并发争用的情况，具体操作步骤如下：

步骤 01 创建 `innodb_monitor` 表，从而打开 InnoDB 的监控功能，命令如下：

```
mysql> create table innodb_monitor(a int)engine=innodb;
Query OK, 0 rows affected (0.01 sec)
```

步骤 02 使用“`SHOW INNODB STATUS`”语句查看 InnoDB 行级锁并发争用的情况，命令如下：

```
mysql> SHOW INNODB STATUS;
=====
140901 11:16:04 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 3 seconds
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 10, signal count 10
```



```

Mutex spin waits 0, rounds 59, OS waits 1
RW-shared spins 18, OS waits 9; RW-excl spins 0, OS waits 0
-----
TRANSACTIONS
-----
Trx id counter 0 115218
Purge done for trx's n:o < 0 114713 undo n:o < 0 0
History list length 17
Total number of lock structs in row lock hash table 5
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 115215, not started, OS thread id 1324
MySQL thread id 2, query id 85 localhost 127.0.0.1 user1
show innodb status
---TRANSACTION 0 115216, ACTIVE 7300 sec, OS thread id 3492
6 lock struct(s), heap size 1024
MySQL thread id 4, query id 76 localhost 127.0.0.1 root
-----
FILE I/O
-----
I/O thread 0 state: wait Windows aio (insert buffer thread)
I/O thread 1 state: wait Windows aio (log thread)
I/O thread 2 state: wait Windows aio (read thread)
I/O thread 3 state: wait Windows aio (write thread)
Pending normal aio reads: 0, aio writes: 0,
  ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
78 OS file reads, 54 OS file writes, 31 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
Ibuf: size 1, free list len 5, seg size 7,
0 inserts, 0 merged recs, 0 merges
Hash table size 70657, used cells 0, node heap has 1 buffer(s)
0.00 hash searches/s, 0.00 non-hash searches/s
---
LOG
---
Log sequence number 0 292071677
Log flushed up to 0 292071677
Last checkpoint at 0 292071677
0 pending log writes, 0 pending chkp writes
22 log i/o's done, 0.00 log i/o's/second
-----
BUFFER POOL AND MEMORY

```

```
-----
Total memory allocated 25627192; in additional pool allocated 1403776
Buffer pool size 1088
Free buffers 1017
Database pages 70
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 68, created 2, written 30
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
-----
ROW OPERATIONS
-----
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread id 1608, state: waiting for server activity
Number of rows inserted 3, updated 0, deleted 0, read 35
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
-----
END OF INNODB MONITOR OUTPUT
=====
```

5.3.3 InnoDB 行级锁的实现方法

InnoDB 行级锁是通过给索引上的索引项加锁来实现的，InnoDB 行级锁只有通过索引条件检索数据，才使用行级锁；否则，InnoDB 将使用表锁。

在不通过索引条件查询的时候，InnoDB 使用的是表锁，而不是行锁。例如在下面的案例中，表 `t_innodb_no_index` 没有索引，此时使用的是表锁定。案例具体操作过程如表 5-12 所示。

表 5-12 InnoDB 表锁定机制的例子

| Session1 | Session2 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| 创建表 <code>t_innodb_no_index</code> ，然后添加数据，命令如下： <pre>mysql> use test Database changed mysql> create table t_innodb_no_index(-> id int, -> name varchar(20) ->)engine=innodb; Query OK, 0 rows affected (0.03 sec)</pre> | |

(续表)

| Session1 | Session2 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>向表 t_innodb_no_index 中插入两条数据, 命令如下:</p> <pre>mysql>insert into t_innodb_no_index values(1,'data1'); Query OK, 1 row affected (0.03 sec) mysql> insert into t_innodb_no_index values(2,'data2'); Query OK, 1 row affected (0.00 sec)</pre> | |
| <p>关闭事务自动提交, 查询数据, 命令如下:</p> <pre>mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from t_innodb_no_index where id = 1; +-----+-----+ id name +-----+-----+ 1 data1 +-----+-----+ 1 row in set (0.06 sec)</pre> | <p>关闭事务自动提交, 查询数据, 命令如下:</p> <pre>mysql> set autocommit=0; Query OK, 0 rows affected (0.00 sec) mysql> select * from t_innodb_no_index where id = 2; +-----+-----+ id name +-----+-----+ 2 data2 +-----+-----+ 1 row in set (0.00 sec)</pre> |
| <p>查询数据, 命令如下:</p> <pre>mysql> select * from t_innodb_no_index where id = 1 for update; +-----+-----+ id name +-----+-----+ 1 data1 +-----+-----+ 1 row in set (0.00 sec)</pre> | |
| | <p>查询数据, 命令如下:</p> <pre>mysql> select * from t_innodb_no_index where id = 2 for update; 等待</pre> |

从结果可以看出, 在表 t_innodb_no_index 没有索引的情况下, InnoDB 使用的只是表级锁。如果使用的表包含索引, 此时 InnoDB 将会锁定符合条件的行。下面通过例子来理解包含索引的表锁定行的例子, 具体过程如表 5-13 所示。

表 5-13 InnoDB 行锁定机制的例子

| Session1 | Session2 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <p>创建表 t_innodb_lock, 命令如下:</p> <pre>mysql> use test; Database changed mysql> create table t_innodb_lock(a int,b varchar(20)) engine=innodb; Query OK, 0 rows affected (0.11 sec)</pre> <p>给表 t_innodb_lock 的 a 列添加索引, 命令如下:</p> <pre>mysql> create index index_innodb_t on t_innodb_lock(a); Query OK, 0 rows affected (0.05 sec) Records: 0 Duplicates: 0 Warnings: 0</pre> <p>给表 t_innodb_lock 的 b 列添加索引, 命令如下:</p> <pre>mysql> create index index_innodb_t2 on t_innodb_lock(b); Query OK, 0 rows affected (0.05 sec) Records: 0 Duplicates: 0 Warnings: 0</pre> <p>插入数据, 命令如下:</p> <pre>mysql>insert into t_innodb_lock values(1,'b1'); Query OK, 1 row affected (0.00 sec) mysql>insert into t_innodb_lock values(2,'b2'); Query OK, 1 row affected (0.00 sec)</pre> | |
| <p>关闭事务自动提交, 查询数据, 命令如下:</p> <pre>mysql> set autocommit=0; Query OK, 0 rows affected (0.06 sec) mysql> select * from t_innodb_lock where a = 1 for update; +-----+-----+ a b +-----+-----+ 1 test +-----+-----+ 1 row in set (0.00 sec)</pre> | <p>关闭事务自动提交, 命令如下:</p> <pre>mysql> set autocommit=0; Query OK, 0 rows affected (0.06 sec)</pre> |

(续表)

| Session1 | Session2 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | <p>查询数据，命令如下：</p> <pre>mysql> select * from t_innodb_lock where a = 2 for update; +-----+-----+ a b +-----+-----+ 2 b2 +-----+-----+ 1 row in set (0.00 sec)</pre> <p>mysql> select * from t_innodb_lock where a = 1 for update;</p> <p>等待</p> |
| <p>提交事务，命令如下：</p> <pre>mysql> commit; Query OK, 0 rows affected (0.00 sec)</pre> | |
| | <p>查询数据，命令如下：</p> <pre>mysql> select * from t_innodb_lock where a = 1 for update; +-----+-----+ a b +-----+-----+ 1 test +-----+-----+ 1 row in set (41.09 sec)</pre> |
| | <p>提交事务，命令如下：</p> <pre>mysql> commit; Query OK, 0 rows affected (0.00 sec)</pre> |
| <p>开始事务，然后更新数据，命令如下：</p> <pre>mysql> start transaction; Query OK, 0 rows affected (0.00 sec) mysql> update t_innodb_lock set b='test' where a = 1; Query OK, 1 row affected (0.13 sec) Rows matched: 1 Changed: 1 Warnings: 0</pre> | <pre>mysql> start transaction; Query OK, 0 rows affected (0.00 sec)</pre> |
| | <p>更新数据，命令如下：</p> <pre>mysql> update t_innodb_lock set b='test' where a = 1;</pre> <p>被阻塞，等待</p> |

(续表)

| Session1 | Session2 |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 提交事务，命令如下： mysql> commit; Query OK, 0 rows affected (0.00 sec) | |
| | 阻塞解除，更新操作执行成功，命令如下： mysql> update t_innodb_lock set b='test2' where a=1; Query OK, 1 row affected (8.42 sec) Rows matched: 1 Changed: 1 Warnings: 0 |

当表中锁定其中的某几行的时候，此时不同的事务可以使用不同的索引锁定不同的行，另外，不论使用主键索引、唯一索引或者普通索引，InnoDB 都会使用行锁来对数据加锁。下面通过例子来理解，操作过程如表 5-14 所示。

表 5-14 InnoDB 行锁定机制的例子

| Session1 | Session2 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 提交事务，命令如下： mysql> commit; Query OK, 0 rows affected (0.00 sec) 开启事务，命令如下： mysql> start transaction; Query OK, 0 rows affected (0.00 sec) | 提交事务，命令如下： mysql> commit; Query OK, 0 rows affected (0.00 sec) 开启事务，命令如下： mysql> start transaction; Query OK, 0 rows affected (0.00 sec) |
| 查询数据，命令如下： mysql> select *from t_innodb_lock where a = 1 for update; +-----+-----+ a b +-----+-----+ 1 test 1 test2 +-----+-----+ 2 rows in set (0.00 sec) | |
| | 使用 b 列的索引访问记录，可以获得该条记录的行锁，命令如下： mysql> select *from t_innodb_lock where a = 2 for update; +-----+-----+ a b +-----+-----+ 2 b2 +-----+-----+ 1 row in set (0.00 sec) |
| | 由于 b=“test”，a=1 的记录已经被 Session1 锁定，查询该记录时会等待获得锁，命令如下： mysql> select *from t_innodb_lock where b = 'test' for update; 等待 |

(续表)

| Session1 | Session2 |
|----------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 提交事务，命令如下： mysql> commit; Query OK, 0 rows affected (0.02 sec) | |
| | 查询 b=“test”的记录，命令如下： mysql> select *from t_innodb_lock where b = 'test' for update; +-----+-----+ a b +-----+-----+ 1 test +-----+-----+ 1 row in set (23.41 sec) |

由此可以看出，Session2 查询其他被 Session1 锁定的行，同样会发生阻塞等待锁资源。

5.3.4 间隙锁（Net-Key 锁）

在更新 InnoDB 存储引擎表中的某个区间数据时，将会锁定这个区间的所有记录，例如，update xxx where id between 1 and 100，此时它会锁住 id 从 1 到 100 之间所有的记录。值得注意的是，在这个区间中假设某条记录并不存在，该条记录也会被锁住，这个时候，如果另外一个 session 往这个表中添加一条记录时，此时必须要等到上一个事务释放锁资源。

InnoDB 使用间隙锁的目的，一方面是为了防止幻读，如果没有添加间隙锁，如果其他事务中添加 id 在 1 到 100 之间的某条记录，此时会发生幻读；另一方面，是为了满足其恢复和赋值的需求。

下面是一个 InnoDB 存储引擎的间隙锁阻塞的例子，操作过程如表 5-15 所示。

表 5-15 InnoDB 存储引擎的间隙锁阻塞例子

| Session1 | Session2 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 查看数据库的隔离级别，命令如下： mysql> select @@tx_isolation; +-----+ @@tx_isolation +-----+ REPEATABLE-READ +-----+ 1 row in set (0.00 sec) | 查看数据库的隔离级别，命令如下： mysql> select @@tx_isolation; +-----+ @@tx_isolation +-----+ REPEATABLE-READ +-----+ 1 row in set (0.00 sec) |
| 关闭事务的自动提交功能，命令如下： mysql> set autocommit = 0; Query OK, 0 rows affected (0.00 sec) | 关闭事务的自动提交功能，命令如下： mysql> set autocommit = 0; Query OK, 0 rows affected (0.00 sec) |

(续表)

| Session1 | Session2 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>对不存在的记录加 for update, 命令如下:</p> <pre>mysql> select max(id) from n_innodb_lock ; +-----+ max(id) +-----+ 1024 +-----+ 1 row in set (0.00 sec)</pre> <p>mysql> select * from n_innodb_lock where id=10000 for update; Empty set (0.00 sec)</p> | |
| | <p>此时, 插入一条新的记录, 命令如下:</p> <pre>mysql>insert into n_innodb_lock(id,ename) values(1026,'test');</pre> <p>等待</p> |
| <p>执行回滚操作, 命令如下:</p> <pre>mysql> rollback; Query OK, 0 rows affected (13.04 sec)</pre> | |
| | <p>由于 Session1 回滚操作释放了间隙锁, 当前成功添加记录, 结果如下:</p> <pre>mysql> insert into n_innodb_lock(id,ename) values(1026,'test');</pre> <p>Query OK, 1 row affected (10.30 sec)</p> |

5.3.5 InnoDB 在不同隔离级别下加锁的差异

在不同的隔离级别下, InnoDB 处理 SQL 语句时所采用的一致性和需要的锁是不同的。

对于 SQL 语句而言, 隔离级别越高, InnoDB 存储引擎给记录添加的锁就越严格, 产生锁冲突的可能性就越高, 对并发的性能影响就越大。因此, 应该尽量使用较低的隔离级别, 以降低并发中锁争用的几率。

对于一些需要使用较高隔离级别的情况, 可以通过如下操作更换隔离级别, 命令如下:

```
mysql> set session transaction isolation level repeatable read;
Query OK, 0 rows affected (0.48 sec)
```

```
mysql> set session transaction isolation level serializable;
Query OK, 0 rows affected (0.00 sec)
```

5.3.6 InnoDB 存储引擎中的死锁

一般情况下，如果 InnoDB 存储引擎发生了死锁状况，通常是一个事务释放锁并回滚，另一个事务获得锁，继续完成事务。但在涉及外部锁，或涉及表锁情况下，InnoDB 并不能完全自动检测到死锁，此时，需要通过设置锁等待时间 (innodb_lock_wait_timeout) 来解决。通常情况下，死锁都是应用设计的问题，通过调整业务流程，事务大小，数据库访问的 SQL 语句，绝大多数死锁都可以避免。下面来看一个 InnoDB 存储引擎发生死锁的例子，操作过程如表 5-16 所示。

表 5-16 InnoDB 存储引擎发生死锁的例子

| Session1 | Session2 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 关闭事务的自动提交功能，命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.05 sec) | 关闭事务的自动提交功能，命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.05 sec) |
| 更新数据，命令如下： mysql> update t_innodb_lock set b='test' where a = 1; Query OK, 1 row affected (0.08 sec) Rows matched: 2 Changed: 1 Warnings: 0 | |
| | 更新数据，命令如下： mysql> update t_innodb_lock set b='test' where a = 2; Query OK, 1 row affected (0.03 sec) Rows matched: 1 Changed: 1 Warnings: 0 |
| 等待 Session2 释放资源，被阻塞，命令如下： mysql> update t_innodb_lock set b='test' where a = 2; | |
| | 更新 a=1 的记录，此时发生死锁，结果如下： mysql> update t_innodb_lock set b='test' where a = 1; ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction |
| 获得锁资源，更新操作成功执行，结果如下： mysql> update t_innodb_lock set b='test' where a = 2; Query OK, 1 row affected (38.66 sec) Rows matched: 1 Changed: 1 Warnings: 0 | |

通常在应用中，在 REPEATABLE-READ 隔离级别下，如果两个线程同时对相同的几率使用了排他锁，在没有符合该条件的情况下，两个线程都会加锁成功。程序发现记录尚不存在，就试图插入一条新记录，如果两个线程都这么做，就会发生死锁。这种情况下，将隔离级别改

成 READ COMMIT，就可以避免死锁。下面通过案例来学习如何操作。

步骤 01 创建测试表 `innodb_dead_lock`，插入测试数据，然后添加索引 `dead_index_id`，命令如下：

```
mysql> create table innodb_dead_lock(
-> id int,
-> data varchar(20)
-> )engine=innodb;
Query OK, 0 rows affected (0.14 sec)

mysql> insert into innodb_dead_lock values(1,'data');
Query OK, 1 row affected (0.05 sec)

mysql> insert into innodb_dead_lock values(2,'data2');
Query OK, 1 row affected (0.00 sec)

mysql> create index dead_index_id on innodb_dead_lock(id);
Query OK, 3 rows affected (0.91 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

步骤 02 接着执行过程如表 5-17 所示。

表 5-17 死锁的执行过程

| Session1 | Session2 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 关闭事务的自动提交功能，命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.05 sec) 查看数据库的隔离级别，命令如下： mysql> select @@tx_isolation; +-----+ @@tx_isolation +-----+ REPEATABLE-READ +-----+ 1 row in set (0.00 sec) 启动事务，命令如下： mysql> start transaction; Query OK, 0 rows affected (0.00 sec) | 关闭事务的自动提交功能，命令如下： mysql> set autocommit=0; Query OK, 0 rows affected (0.05 sec) 查看数据库的隔离级别，命令如下： mysql> select @@tx_isolation; +-----+ @@tx_isolation +-----+ REPEATABLE-READ +-----+ 1 row in set (0.00 sec) 启动事务，命令如下： mysql> start transaction; Query OK, 0 rows affected (0.00 sec) |
| 查询数据，命令如下： mysql> select * from innodb_dead_lock where id=102 for update; Empty set (0.00 sec) | 查询数据，命令如下： mysql> select * from innodb_dead_lock where id=102 for update; Empty set (0.00 sec) |

(续表)

| Session1 | Session2 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 插入数据，命令如下： mysql> insert into innodb_dead_lock values(102,'data102'); 等待 | |
| | 其他 session 已经对记录进行了更新，此时再插入 此条记录就会发生死锁并退出，结果如下： mysql> insert into innodb_dead_lock values(102,'data102'); ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction |
| 由于其他 session 已经退出，当前 session 可以获得 锁并成功插入记录，结果如下： mysql> insert into innodb_dead_lock values(102,'data102'); Query OK, 1 row affected (29.95 sec) | |

5.3.7 InnoDB 行级锁优化建议

InnoDB 存储引擎由于实现了行级锁，很显然，在锁定方面行级锁的颗粒更小，实现更为复杂，所带来的性能损耗也比表级锁更高；但是，InnoDB 行锁在并发性能上远远要高于表级锁定，当系统并发量较高的时候，InnoDB 的整体性能和 MyISAM 相比优势就比较明显了，所以在选择使用哪种锁的时候，应该考虑到该应用是否有很大的并发量，要想合理使用 InnoDB 的行锁，应该扬长避短，尽量做到以下几点：

- (1) 控制事务的大小，减少锁定的资源量和锁定时间长度。
- (2) 让所有的数据检索都通过索引来完成，从而避免因为无法通过索引加锁而升级为表级锁定。
- (3) 减少基于范围的数据检索过滤条件，避免因为间隙锁带来的负面影响而锁定了不该锁定的记录。
- (4) 在业务环节允许的情况下，尽量使用较低级别的事务隔离，以减少因为事务隔离级别锁带来的附加成本。
- (5) 合理使用索引，让 InnoDB 在索引上面加锁的时候更加准确。
- (6) 在应用中，尽可能按照相同的访问顺序来访问，防止产生死锁。
- (7) 在同一个事务中，尽可能做到一次锁定所需的所有资源，减少产生死锁的几率。
- (8) 对于容易产生死锁的业务，可以放弃使用 InnoDB 行级锁定，尝试使用表级锁定来减少死锁产生的概率。
- (9) 不要申请超过实际需要的锁级别。

5.4 小结

本章对 MySQL 中使用最为频繁的存储引擎 MyISAM 和 InnoDB 各自的锁定机制进行了分析，重点讲解了 MyISAM 表级锁和 InnoDB 行级锁的实现特点。

对于 MyISAM 表级锁主要讲解到共享锁(S)、排他锁(X)，以及排他锁之间互斥特性和表锁并发争用的问题，由于表锁的颗粒度比较大，因此，如果更新操作比较多，频繁占用表资源而导致引用的性能下降。

对于 InnoDB 行级锁主要讲解到行级加锁方式，以及 InnoDB 间隙锁(Next-key)的机制，InnoDB 存储引擎因为支持事务的 ACID 特性，在不同的隔离级别下，InnoDB 锁机制和一致性策略不一样。另外讲解了 InnoDB 产生死锁的情况，以及如何避免死锁。

第 6 章

使用MySQL Workbench 管理数据库

MySQL 数据库的管理工具很多，MySQL Workbench 无疑是其中非常好的管理工具之一，MySQL Workbench 5.2 提供了图形化界面下的数据库基本的管理，数据库建立物理模型，以及通过物理模型转换成执行的 SQL 脚本，另外 MySQL Workbench 5.2 提供了对 MySQL 数据库性能的监控，用户的管理，以及备份和还原数据库数据。

本章中通过使用 MySQL Workbench 5.2 管理 MySQL 工具，来对数据库的一些基本操作进行阐述，希望对读者朋友有一定的帮助。

6.1 MySQL Workbench 简介

MySQL Workbench 是 MySQL 图形界面管理工具，跟其他数据库图形界面管理工具一样，该工具可以对数据库进行创建数据库表、增加数据库表、删除数据库和修改数据库等操作。

6.1.1 MySQL Workbench 的概述

MySQL Workbench 是一款专门为用户提供了用于创建、修改、执行和优化 SQL 的可视化工具，开发人员可以很轻松地管理数据库。该工具为开发者提供一整套可视化的用于创建、编辑和管理 SQL 查询和管理数据库连接的功能。在可视化 SQL 编辑工作模式下，用户创建表，删除表，修改表信息等只需要在简单的可编辑列表中完成。

MySQL Workbench 在数据库管理这块也提供了可视化的操作，包括管理用户，授予和收回用户权限等；并且在数据库管理中，可以查看到数据库的状态，其中包括数据库中开启多少个客户端，数据库缓存的大小以及管理数据库日志等信息。

MySQL Workbench 在数据库管理中导入、导出数据库信息这方面提供了比较方便的操作，可以参考本章节的 6.3.2 小节的介绍。

另外，MySQL Workbench 提供了数据库建模管理设计，可以说是著名的数据库设计工具 DBDesigner4 的继承者，可以设计和创建新的数据库物理模型；MySQL Workbench 可以说是下一代数据库可视化设计管理工具的佼佼者，目前 MySQL Workbench 提供了开源和商业化两个版本，同时支持 Windows 和 Linux 系统。

6.1.2 MySQL Workbench 的优势

目前流行的 MySQL GUI Tools 有很多, 常见的有 MySQL Query Brower、MySQL Administrator 和 MySQL System Tray Monitor 等工具, MySQL Workbench 跟大部分 MySQL 管理工具一样, 提供了 MySQL 语法校验, 可视化操作下创建数据 SCHEMA、表和视图等数据库对象。

除此之外, MySQL Workbench 管理工具在 MySQL 管理方面独树一帜, 提供了对数据库服务的启动/停止的管理, 以及查看用户连接次数和数据库健康状况。

用户通常认为 MySQL Workbench 是一个 MySQL 数据库 ER 模型设计的工具, 可以说是专门为 MySQL 数据库提供的数据库设计工具, 用户使用 MySQL Workbench 可以很容易地设计、编辑数据库 ER 模型。这一功能可以说是 MySQL Workbench 的一大亮点。

6.1.3 MySQL Workbench 的安装

了解 MySQL Workbench 的功能后, 下面需要在 MySQL 的官方网站下载 Workbench 软件, 下载地址是: <http://www.mysql.com/downloads/workbench/>, 如图 6-1 所示。

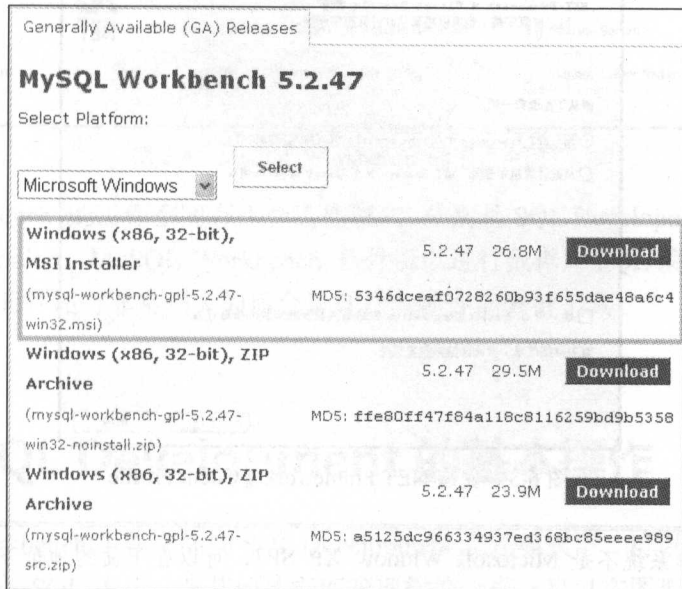


图 6-1 MySQL Workbench 下载页面

这里下载 `mysql-workbench-gpl-5.2.47-win32.msi`, 值得注意的是, 只有登录该网站之后才能下载, 下载完成之后就可以安装该软件。

提示

第一次安装 MySQL Workbench 时, 需要保证安装环境, 包括 Window XP with SP3 系统和 .NET Framework 4 Client Profile, 如果没有上述安装环境, 将会提示以下错误信息。

The operation system is not adequate for running MySQL Workbench 5.2 CE.
You need Windows XP with SP3 and .Net 4.0 Client Profile install.

可以在 <http://www.microsoft.com/zh-cn/download/details.aspx?id=24872> 下载 .NET Framework 4 Client Profile, 如图 6-2 所示。

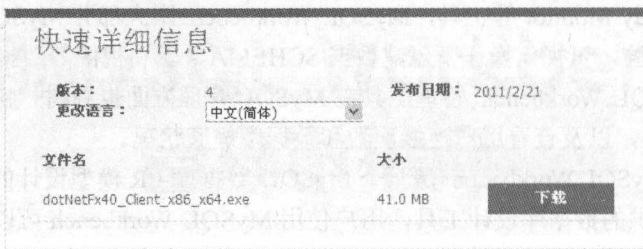


图 6-2 下载.NET Framework 4 Client Profile 页面

下载完成后, 即可直接双击安装程序, 开始安装.NET Framework 4 Client Profile 程序, 如图 6-3 所示。按照提示一直单击【下一步】按钮, 即可完成安装过程。

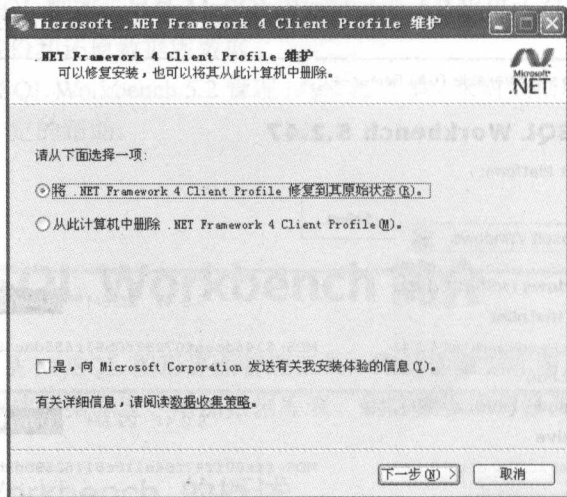


图 6-3 安装 NET Framework 4 Client Profile

提示

如果操作系统不是 Microsoft Window XP SP3, 可以在下面的地址下载并安装 Microsoft Window XP SP3, 下载地址: <http://down.tech.sina.com.cn/content/37717.html>。

接下来我们开始安装 mysql-workbench-gpl-5.2.47-win32.msi 程序, 安装过程比较简单, 按照提示一直单击【下一步】按钮安装即可, 安装完成之后, 打开 MySQL Workbench 工作空间, 如图 6-4 所示。

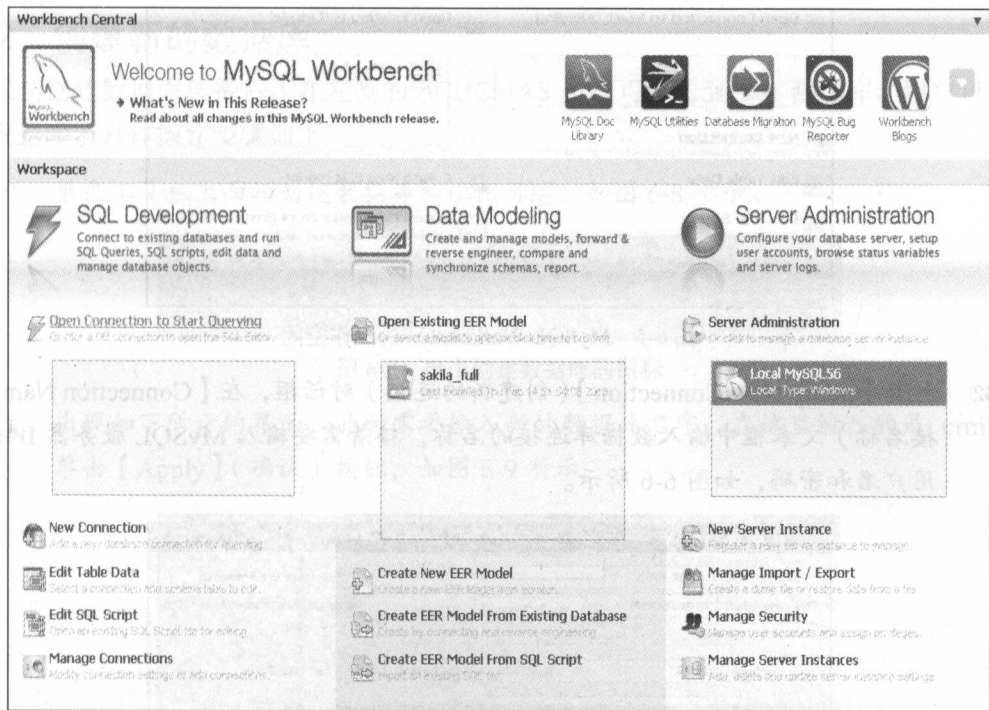


图 6-4 MySQL Workbench 的工作空间

MySQL Workbench 的工作空间有 3 个工作模式，分别是 SQL Development、Data Modeling 和 Server Administration。MySQL Workbench 软件可以进行数据库 SQL 编辑、错误提示、可视化数据库设计，此外还有非常强大的后台管理功能和性能分析工具。

6.2 SQL Development 的基本操作

在 MySQL Workbench 工作空间的 SQL Development 工作模式下，可以创建一个新的数据库连接，编辑并运行 SQL 语句，跟其他数据库管理软件一样，可以在图形化界面中管理数据库表的基本信息，本小节将学习如何通过 MySQL Workbench 在图形化界面中创建并管理数据库信息。

6.2.1 创建数据库连接

MySQL Workbench 工作空间下对数据库数据进行管理之前，需要先创建数据库连接，具体操作步骤如下。

步骤 01 在 MySQL Workbench 工作空间中，单击【New Connection】按钮，如图 6-5 所示。

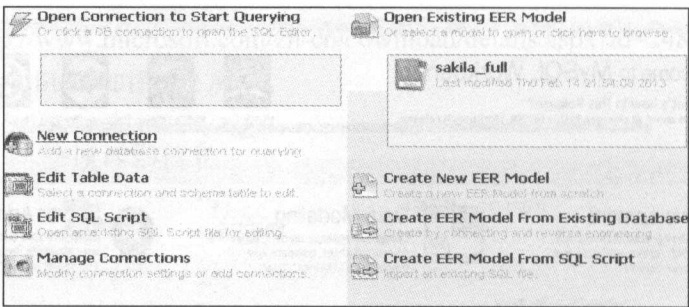


图 6-5 MySQL Workbench 的工作空间

步骤 02 打开【Setup New Connection】(创建新的连接)对话框，在【Connection Name】(连接名称)文本框中输入数据库连接的名称，接着需要输入 MySQL 服务器 IP 地址、用户名和密码，如图 6-6 所示。

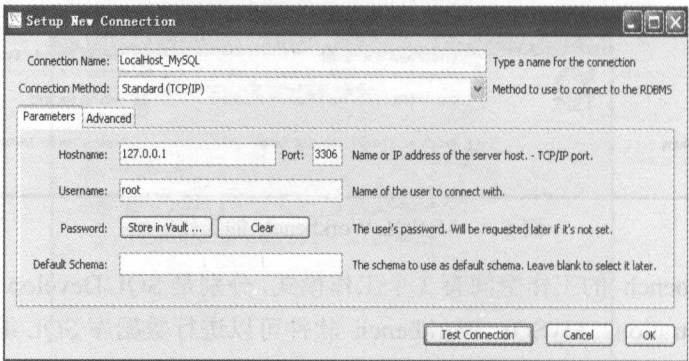


图 6-6 【Setup New Connection】对话框

步骤 03 单击【OK】按钮，连接 MySQL 数据后的界面如图 6-7 所示。左侧展示的是 test 库中的 Tables、Views 以及 Routines。Query 1 窗口用来执行 SQL 语句的窗口，右侧的 SQL Additions 窗口是用来帮助用户写 SQL 语句的提示窗口。

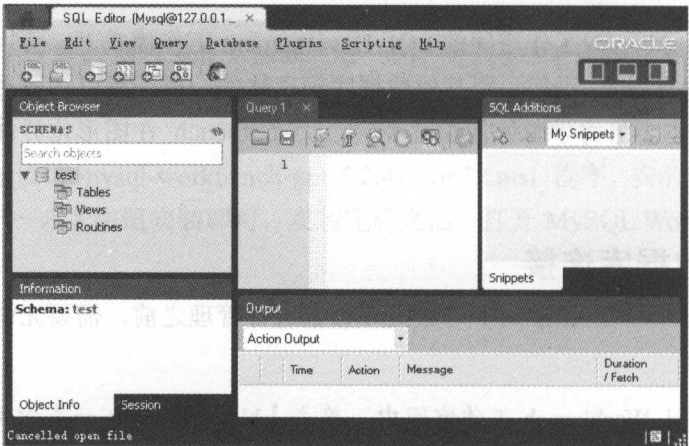



图 6-7 MySQL Workbench 连接数据库

6.2.2 创建新的数据库

成功创建数据库连接后，在左侧的 SCHEMAS 下面可以看到 test 数据库。用户可以创建新的数据库，具体操作步骤如下。

步骤 01 单击工具栏上面的创建数据库的小图标，如图 6-8 所示。

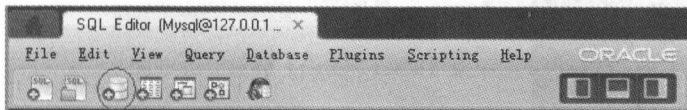


图 6-8 单击创建数据库的图标

步骤 02 出现如下所示的界面，此时需要输入新的数据库名字，在这里输入的是 crm，然后单击【Apply】（确认）按钮，如图 6-9 所示。

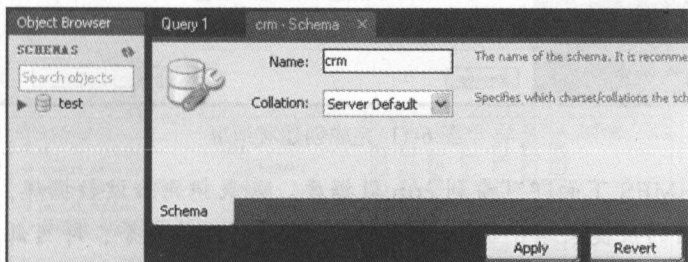


图 6-9 创建新的数据库

步骤 03 弹出一个新的界面，即可看到创建数据库的语句，然后单击【Apply】（确认）按钮，如图 6-10 所示。

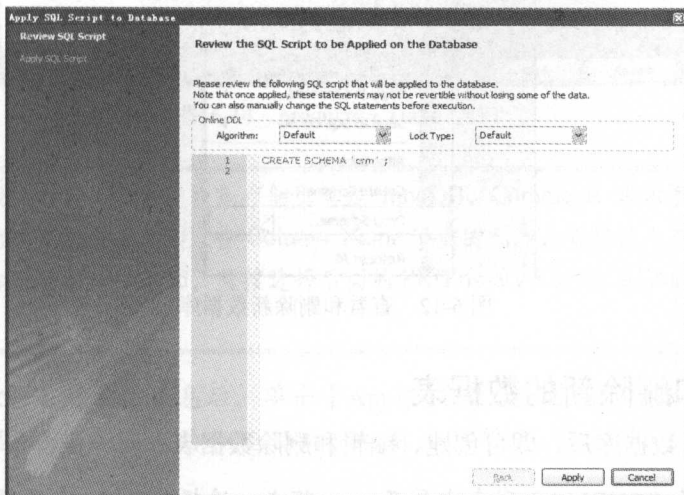


图 6-10 创建数据库的语句

步骤 04 弹出一个新的界面，然后单击【Finish】（完成）按钮，完成创建数据库的操作，如图 6-11 所示。

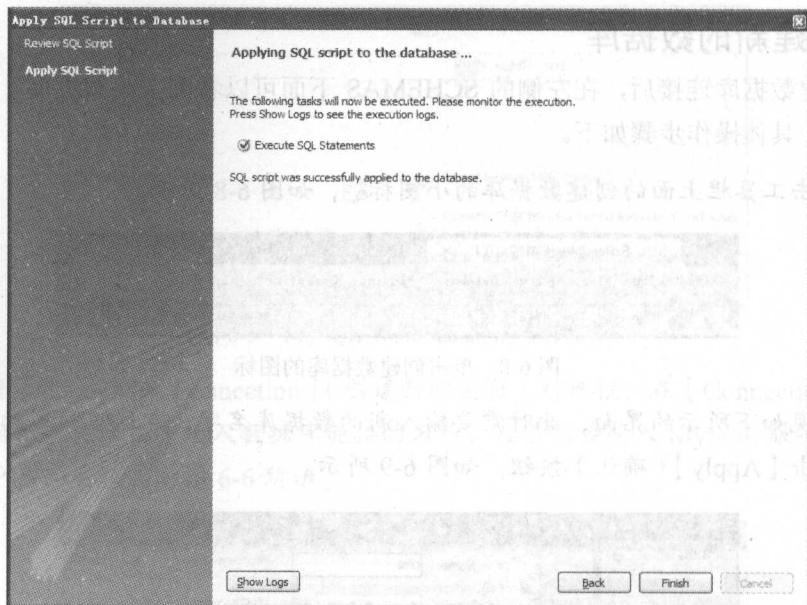


图 6-11 完成创建对话框

步骤 05 在 SCHEMAS 下面即可看到 `crm` 数据库。如果想删除该数据库，可以选择后右击，并在弹出的快捷菜单中选择【Drop Schema】菜单命令，即可删除 `crm`，如图 6-12 所示。

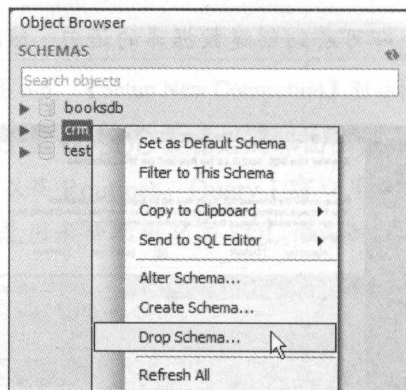


图 6-12 查看和删除新数据库

6.2.3 创建和删除新的数据表

成功创建 `crm` 数据库后，即可创建、编辑和删除数据表。具体操作步骤如下。

步骤 01 在左侧的 SCHEMAS 列表中展开 `crm` 节点，选择【Tables】选项，右击并在弹出的快捷菜单中选择【Create Table】菜单命令，如图 6-13 所示。

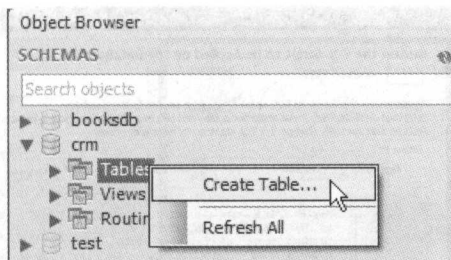


图 6-13 选择【Create Table】菜单命令

步骤 02 在右侧弹出的 products-Table 窗口中可以添加表的信息，比如表的名称和表中各列的相关信息，如图 6-14 所示。

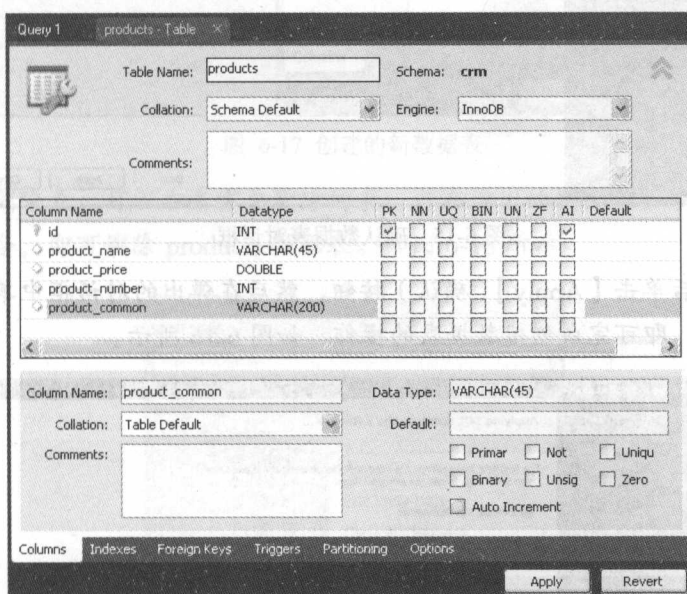


图 6-14 products - Table 窗口

提示

在创建表的时候，默认的数据存储引擎是 InnoDB，Comments 表示表述信息，增加该表各字段信息可以鼠标直接点击 Column Name 下面的列表，直接输入表列的名称和数据类型，比如需要创建主键 id，直接选择后面的 PK(Primary Key)和 AI(Auto Increment)复选框即可。

步骤 03 设置完数据表的基本信息后，单击【Apply】(确认)按钮。弹出一个确认对话框，该对话框上有自动生成的 SQL 语句，如图 6-15 所示。

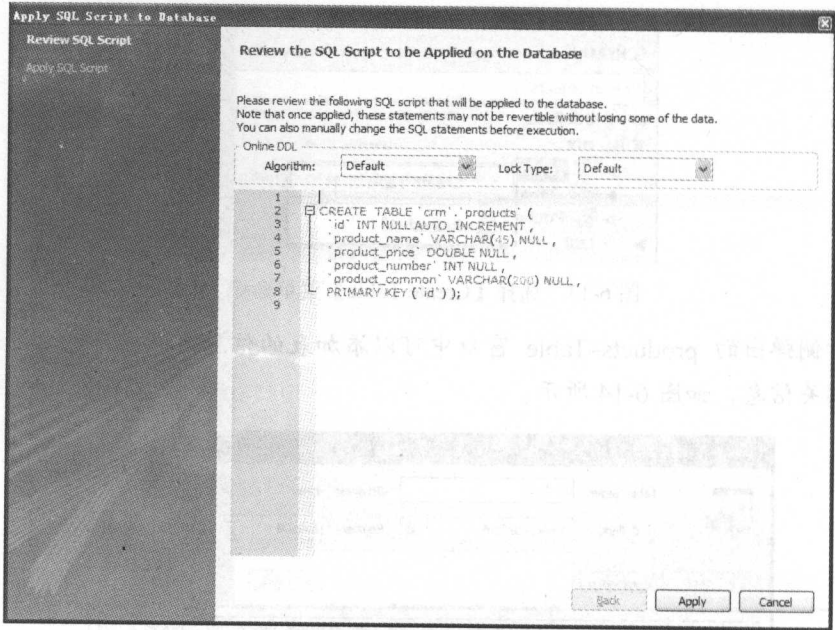


图 6-15 确认数据表对话框

步骤 04 确定无误后单击【Apply】（确认）按钮，然后在弹出的对话框中单击【Finish】（完成）按钮，即可完成创建数据表的操作，如图 6-16 所示。

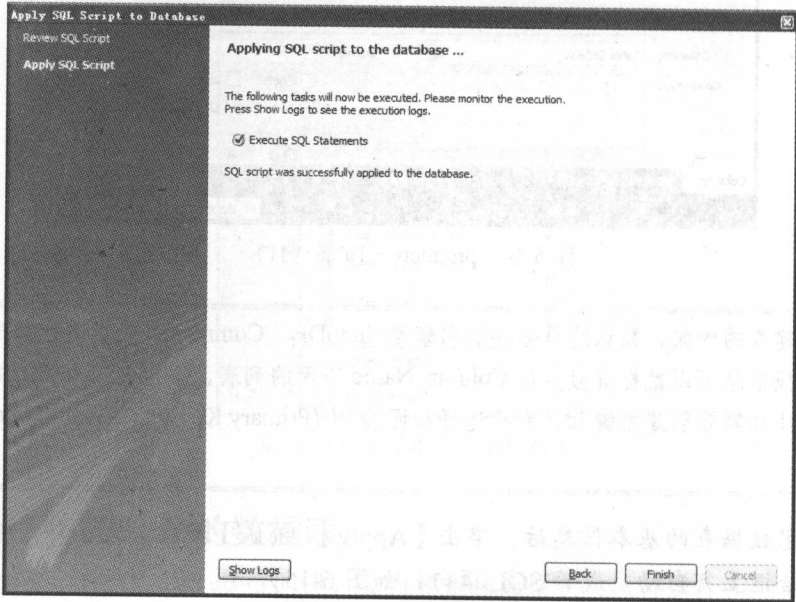


图 6-16 完成创建数据表对话框

步骤 05 创建完表 products 之后，会在 Tables 节点下面展现出来，如图 6-17 所示。

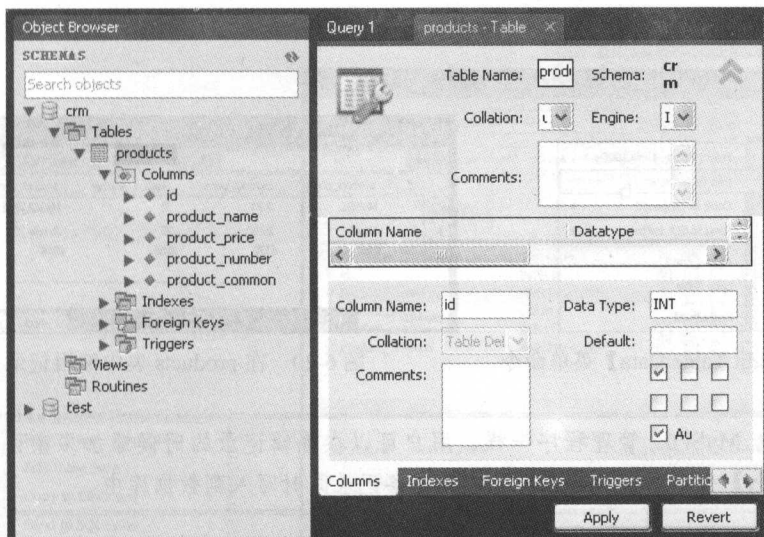


图 6-17 创建的新数据表

步骤 06 如果想删除表，可以右击需要删除的表，并在弹出的快捷菜单中选择【Drop Table】菜单命令，即可删除 products 数据表。如图 6-18 所示。

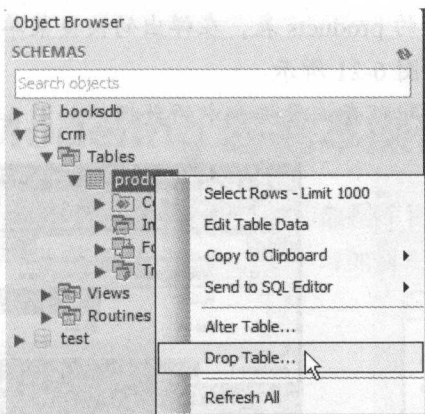


图 6-18 删除数据表

6.2.4 添加、修改表记录

用户可以通过执行添加表记录的 SQL 语句来添加数据，也可以在 Query1 的窗口执行 SQL 语句，这种方法效率不高。下面介绍的是 MySQL Workbench 在图形界面下对数据库表的维护，这种操作方式是非常方便的，具体操作步骤如下。

步骤 01 右击节点 Tables 下面的 products 表，在弹出的快捷菜单中选择【Edit Table Data】菜单命令，如图 6-19 所示。

步骤 02 用户即可在右侧弹出的窗口中编辑数据表中的数据，如图 6-20 所示。编辑完成后，单击【Apply】（确认）按钮即可。

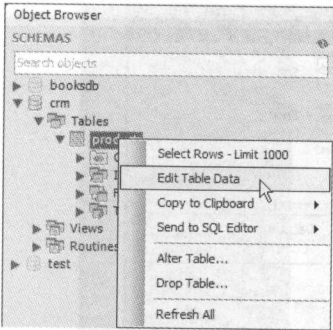


图 6-19 选择【Edit Table Data】菜单命令

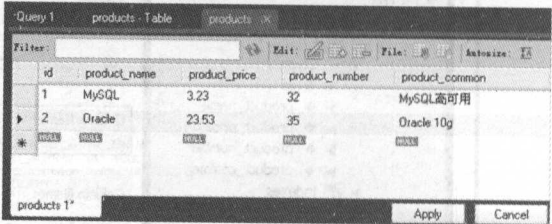


图 6-20 在 products 表中编辑记录

提示

跟其他 MySQL 管理程序一样，用户可以在编辑记录的时候添加多条记录，然后单击【Apply】（确认）按钮，会直接将多条记录同时写入到数据库中。

6.2.5 查询表记录

添加了若干条记录到数据库的 products 表中，下面即可查询数据库的数据。具体操作步骤如下。

- 步骤 01 右击节点 Tables 下面的 products 表，在弹出的快捷菜单中选择【Select Rows - Limit 1000】菜单命令，如图 6-21 所示。
- 步骤 02 在右侧打开的窗口中即可查询数据表中的数据。如图 6-22 所示。

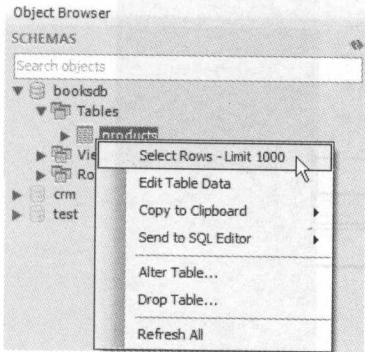


图 6-21 选择需要查询的数据表

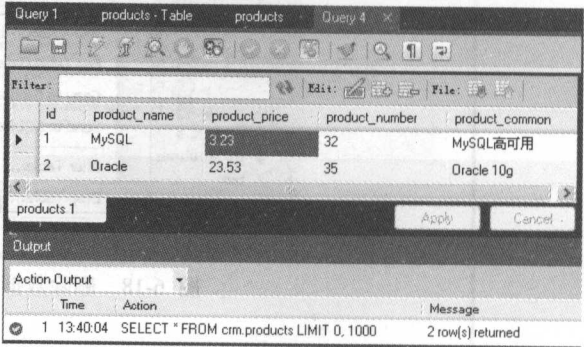


图 6-22 查询 products 表的记录

6.2.6 修改表结构

根据工作的实际需求，用户可以修改表的结构。具体的操作步骤如下。

- 步骤 01 右击节点 Tables 下面的 products 表，在弹出的快捷菜单中选择【Alter Table】菜单命令，如图 6-23 所示。
- 步骤 02 在右侧打开的窗口中即可修改数据表中的结构，如图 6-24 所示。在修改某列数据类型时候，比如需要将 VARCHAR(45)改成 VARCHAR(200)，可以直接在下面的对话

框中修改成 VARCHAR(200)。

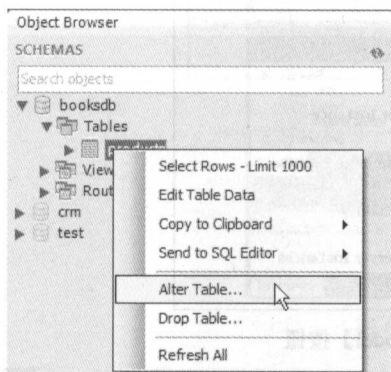


图 6-23 选择【Alter Table】菜单命令

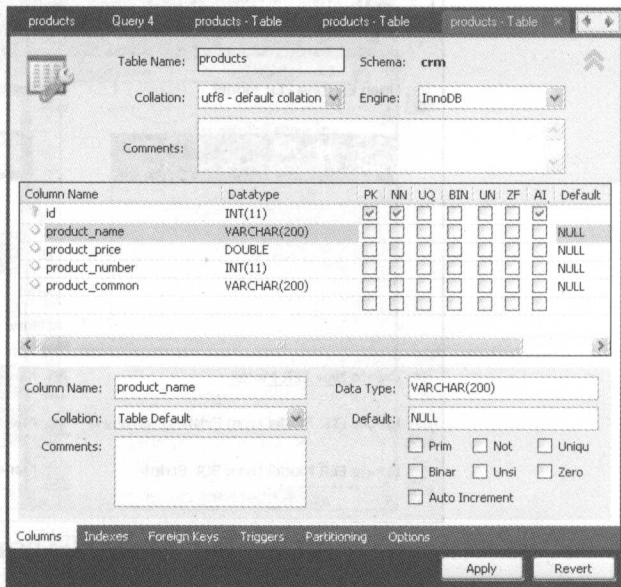


图 6-24 修改 products 表的结构

6.3 Data Modeling 的基本操作

MySQL Workbench 为 MySQL 提供了 ER 图设计和数据库物理建模,用户可以建立 ER 图,可以使用它生成数据库对象。本小节通过一个例子来学习如何在 MySQL Workbench 中建立物理模型,然后导出 SQL 语句。

6.3.1 建立 ER 模型

以购物系统为例,建立数据的物理模型。购物系统中包括以下几个物理模型,用户、商品和订单等。用户和订单之间是 1 个用户对映 1 个订单,是 1:1 的关系,而订单和商品之间的关系是一对多 (1:m) 的关系。具体的操作步骤如下。

步骤 01 在 MySQL Workbench 的 Data Modeling 工作模式下,单击【Create New EER Model】按钮,如图 6-25 所示。

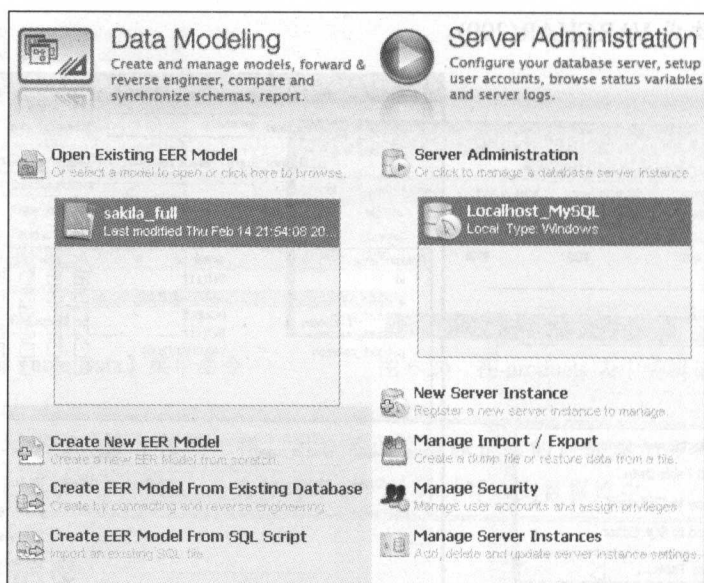





图 6-25 单击【Create New EER Model】按钮

步骤 02 打开 Model Overview 窗口，在该窗口下首先需要创建新的 SCHEMA，如图 6-26 所示。单击【新增】按钮 ，此时会产生一个新的 SCHEMA，然后输入 SCHEMA 的名字 shopping，输入完成后，单击【关闭】按钮 ，最后单击【Add Diagram】按钮 .

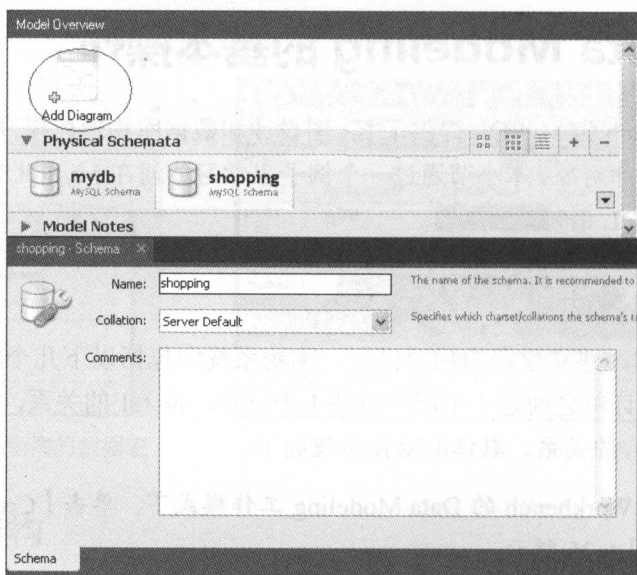



图 6-26 创建一个新的模型

步骤 03 打开 EER Diagram 窗口，单击创建表按钮 ，在②的地方改成前面所添加的 shopping，如图 6-27 所示。

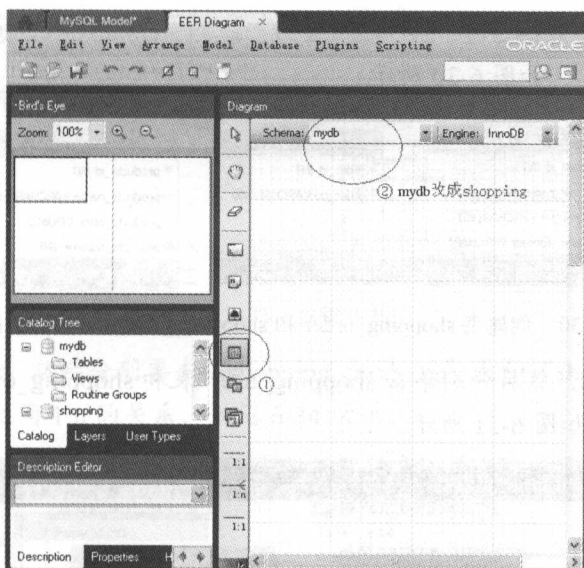


图 6-27 开始添加数据库物理模型

步骤 04 接下来单击窗口空白的地方，此时会创建一个 Table1，如图 6-28 所示。

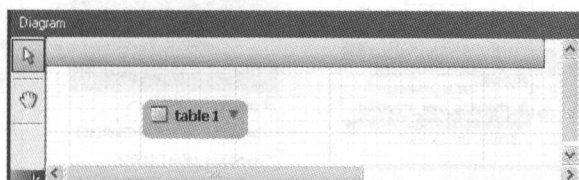


图 6-28 创建 Table1

步骤 05 双击 Table1 之后会出现编辑 Table1 的窗口，如图 6-29 所示。窗口下方出现【shopping_user-Table】选项，在该页面中修改 Table Name 为 shopping_user，然后添加了主键 user_id，以及添加各个列的相关信息。添加列之后，可以选择数据类型和约束。

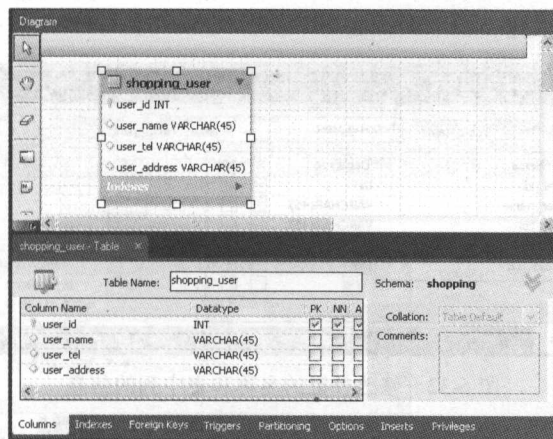


图 6-29 编辑 Table1

步骤 06 使用相同的方法创建两个表 `shopping_order` 和 `shopping_products`，然后给表添加各个列的相关信息，如图 6-30 所示。



图 6-30 创建表 `shopping_order` 和 `shopping_products` 并添加信息

步骤 07 单击 1:1 按钮，然后分别单击 `shopping_user` 表和 `shopping_order` 表，使两个表建立 1:1 的关系，如图 6-31 所示。

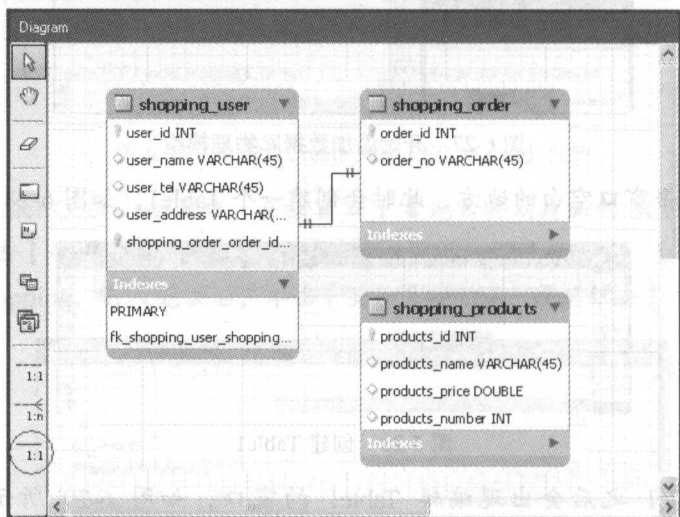


图 6-31 使 `shopping_order` 和 `shopping_products` 建立 1:1 的关系

步骤 08 修改刚刚建立的一个外键的名字，此时双击表“`shopping_user`”，然后修改表中列的信息，如图 6-32 所示。

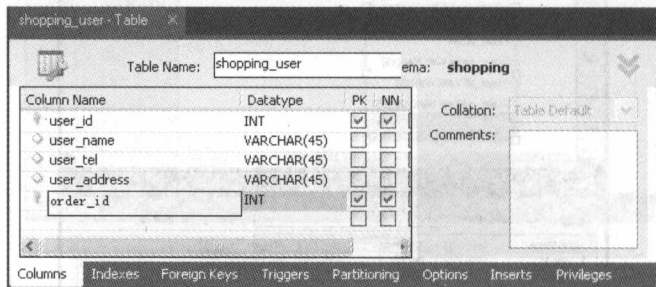


图 6-32 修改外键的名称和表中列的信息

步骤 09 切换到【Foreign Key】选项下面，可以很清楚地看到 `shopping_user` 表的外键直接引用的是 `shopping_order` 表。用户表和订单表是 1:1 的关系，已经设计完成。如图 6-33 所示。

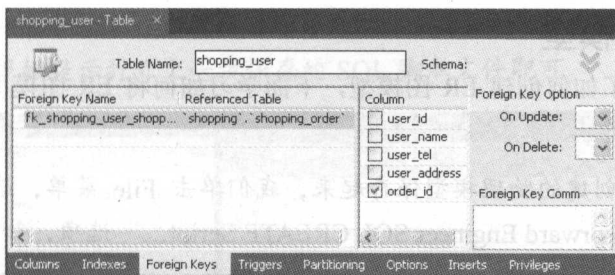


图 6-33 外键连接

步骤 10 单击 1:n 按钮，然后分别单击“shopping_products”表和“shopping_order”表，此时会建立两表之间 1:n 的关系，如图 6-34 所示。

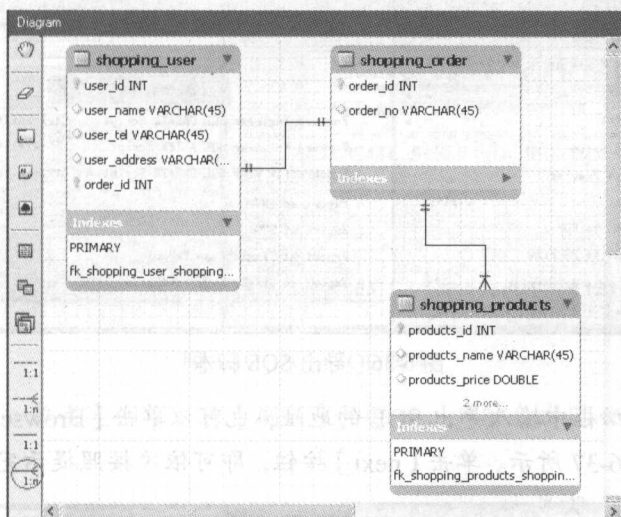


图 6-34 建立 1:n 的关系

步骤 11 双击“shopping_products”表会显示如下界面，用户可以修改该表的列的相关信息，此时已经建立了一个 shopping_order_order_id 的外键，如图 6-35 所示。

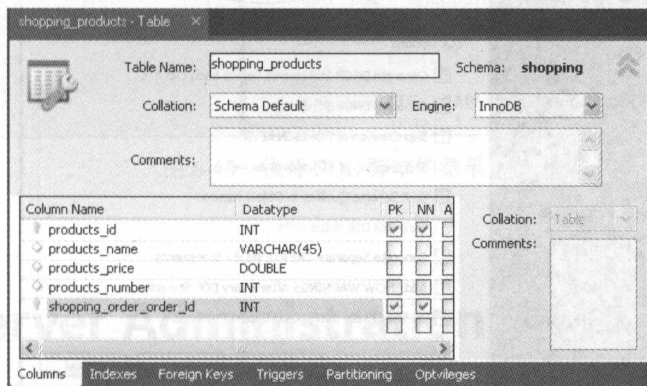


图 6-35 建立 1:n 的关系

6.3.2 导入 ER 模型

上一小节中讲述了如何创建 ER 图模型，下面学习如何将 ER 图模型直接转换成 SQL 脚本，具体操作步骤如下。

步骤 01 下面可以将创建的物理模型保存起来，我们单击 File 菜单，选择“Export”选项，之后选择“Forward Engineer SQL CREATE Script...”选项，如图 6-36 所示。

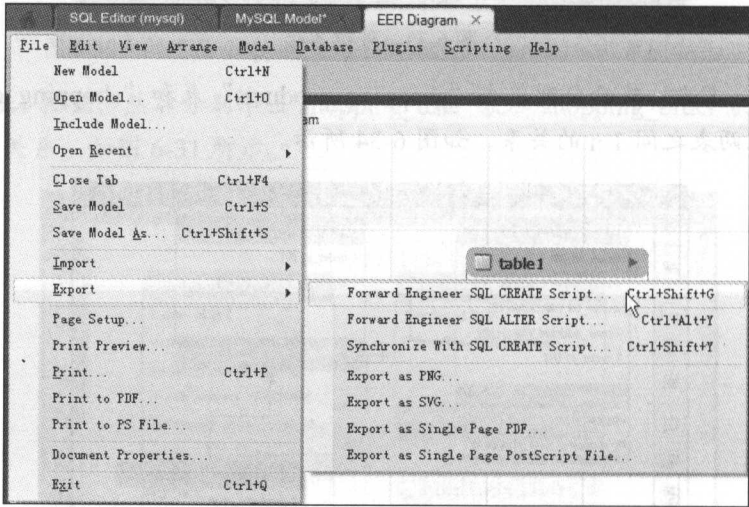


图 6-36 导出 SQL 脚本

步骤 02 在打开的对话框中输入导出 SQL 的地址，也可以单击【Browse】按钮后选择导出的路径，如图 6-37 所示。单击【next】按钮，即可依次按照提示完成导出脚本。至此，完成导出 SQL 脚本。

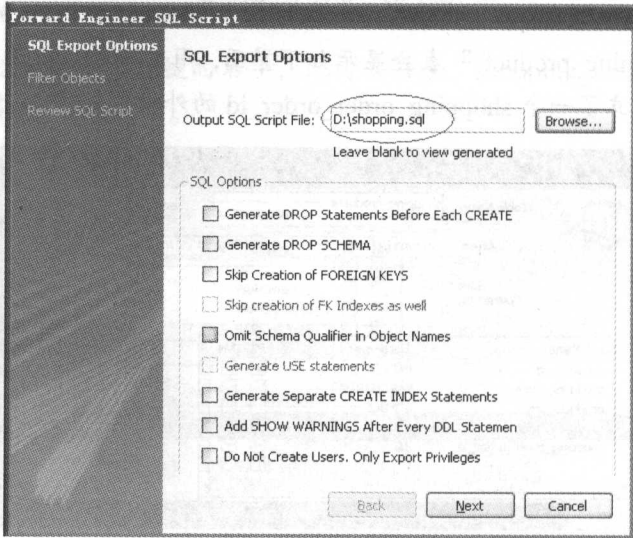



图 6-37 导出 SQL 脚本

步骤 03 在 SQL Development 工作模式下可以导入脚本，单击【导入文件】按钮，如图 6-38 所示。然后根据提示打开上一步保存的 SQL 脚本文件即可。

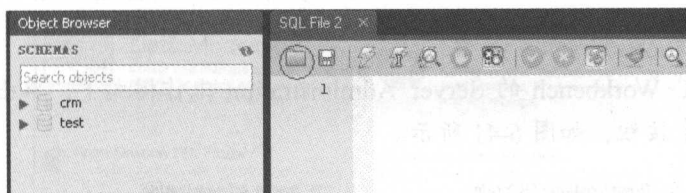


图 6-38 导入 SQL 脚本

步骤 04 单击【执行】按钮，即可开始自动执行脚本，如图 6-39 所示。

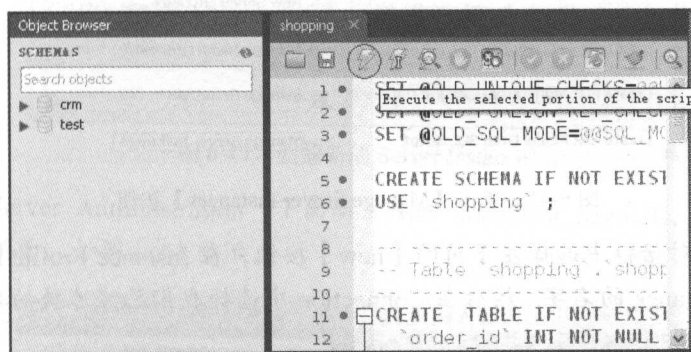



图 6-39 执行 SQL 脚本

步骤 05 执行脚本后，单击左侧窗口的【刷新】按钮，此时会出现 shopping 数据库，如图 6-40 所示。

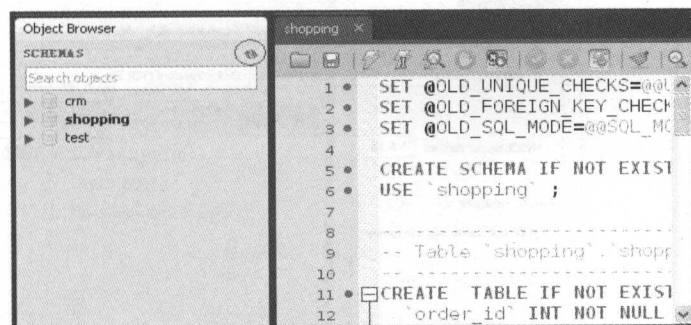


图 6-40 执行 SQL 脚本的结果

6.4 Server Administration 的基本操作

MySQL Workbench 工作空间的 Server Administration 工作模式下数据库管理中提供了数据库状态监控、服务器开关，以及数据库日志管理、用户授权管理与备份和还原数据库信息。

6.4.1 管理 MySQL 用户

MySQL Workbench 数据库管理中提供了对 MySQL 用户的管理，用户可以在图形界面下很方便地管理数据用户。具体操作步骤如下。

步骤 01 在 MySQL Workbench 的 Server Administration 工作模式下，单击【Manage Server Instances】按钮，如图 6-41 所示。

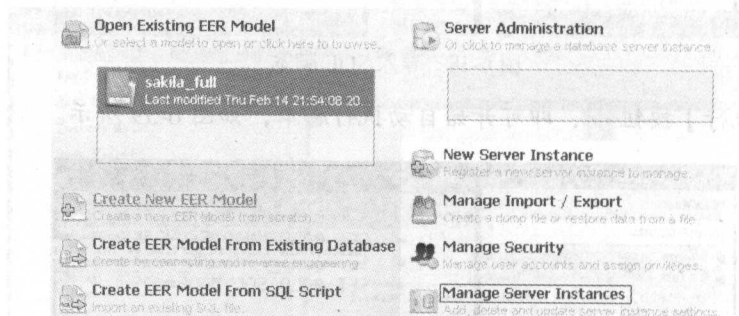


图 6-41 单击【Manage Server Instances】按钮

步骤 02 在弹出的对话框中选择左下角的【new】按钮，在 Instance Profile Name 选项中输入 Server Instance 的名字，然后在 Connection 中选择我们已经连接的数据库实例，如图 6-42 所示。

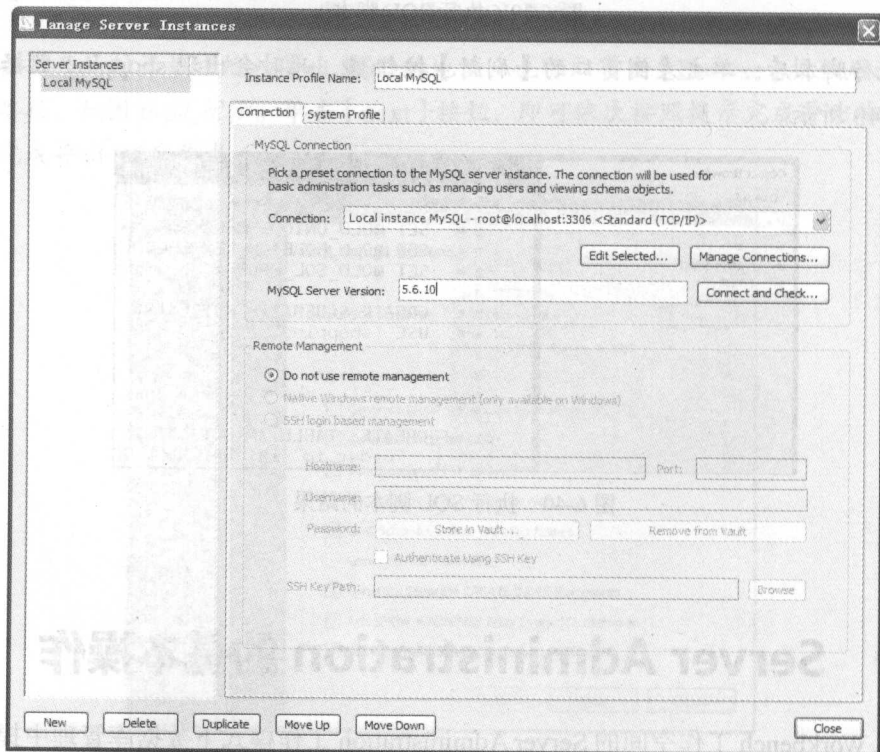


图 6-42 设置【Manage Server Instances】对话框

步骤 03 单击【close】按钮后，在 Server Administration 下面就会出现可用的 Server Instance，如图 6-43 所示。

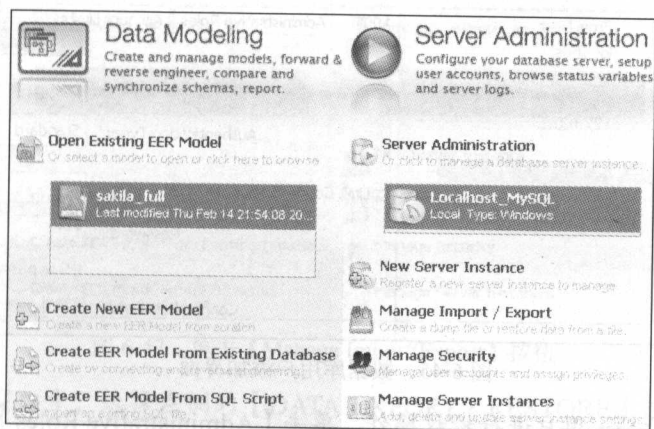


图 6-43 创建新的 Server Instance

步骤 04 双击“Server Administration”下面新出现的 Localhost_MySQL，此时会出现数据库管理的界面，如图 6-44 所示。

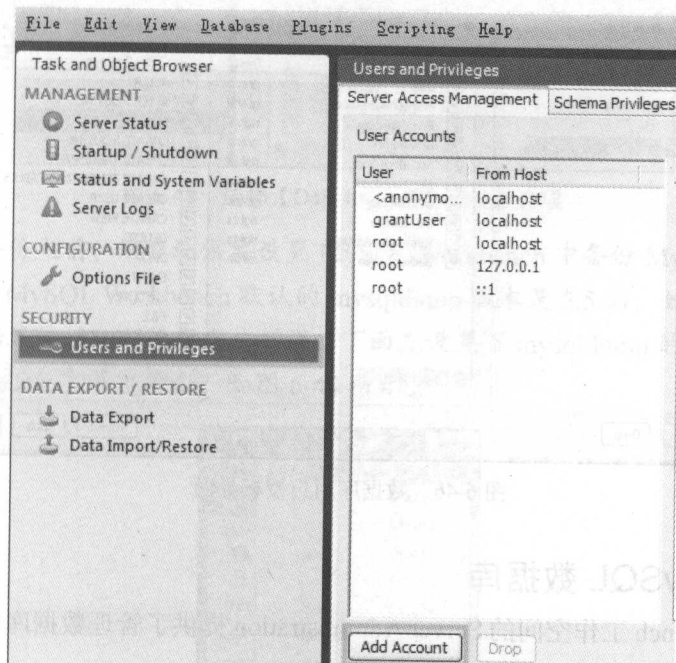


图 6-44 数据库管理界面

步骤 05 单击界面左侧的【Users and Privileges】选项，然后单击右侧界面下方的【Add Account】按钮，接着开始添加 test 用户，如图 6-45 所示。输入用户名和密码，其中 Limit Connectivity to Hosts Matching 表示限制连接 MySQL 服务器的远程地址。%表示不受限制。

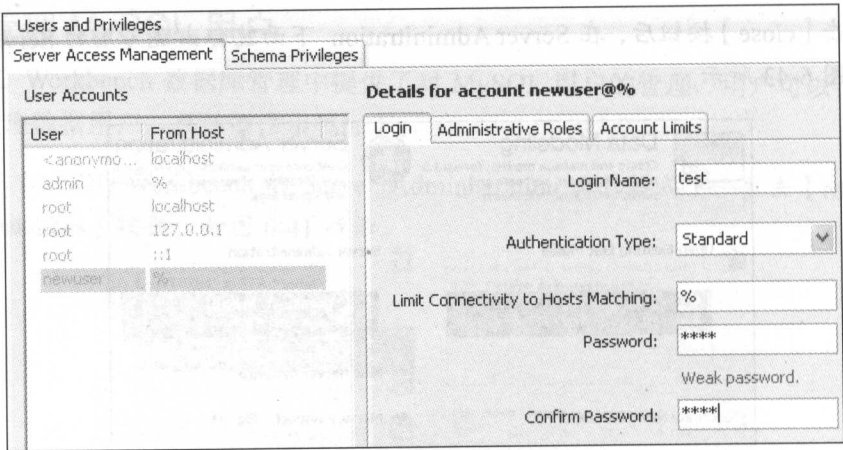


图 6-45 数据库用户添加界面

步骤 06 下面开始对 test 用户进行授权操作。单击【Administrative Roles】选项，选择需要的角色和权限，单击【Apply】按钮之后，test 用户创建成功。如图 6-46 所示。

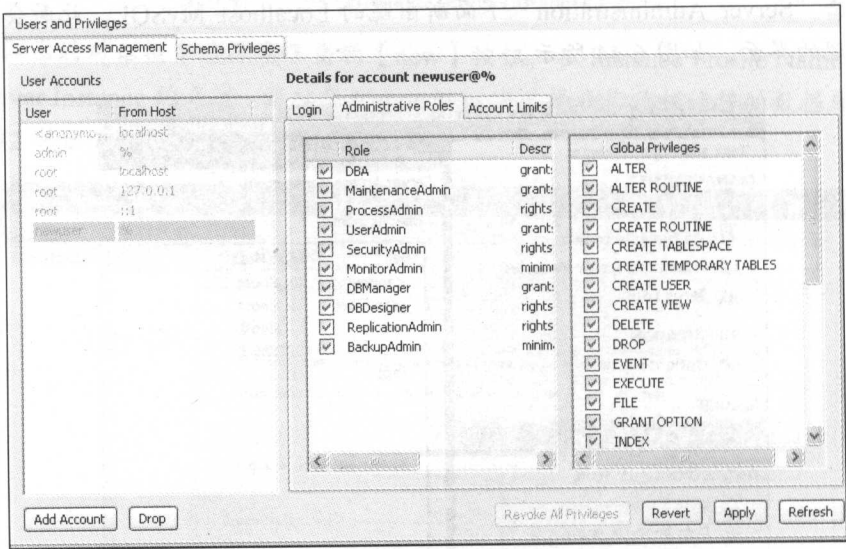


图 6-46 数据库用户授权界面

6.4.2 备份 MySQL 数据库

MySQL Workbench 工作空间的 Server Administration 提供了管理数据库的备份和还原的功能。具体操作步骤如下。

步骤 01 在 MySQL Workbench 的 Server Administration 工作模式下，单击【Manage Import/Export】按钮，如图 6-47 所示。

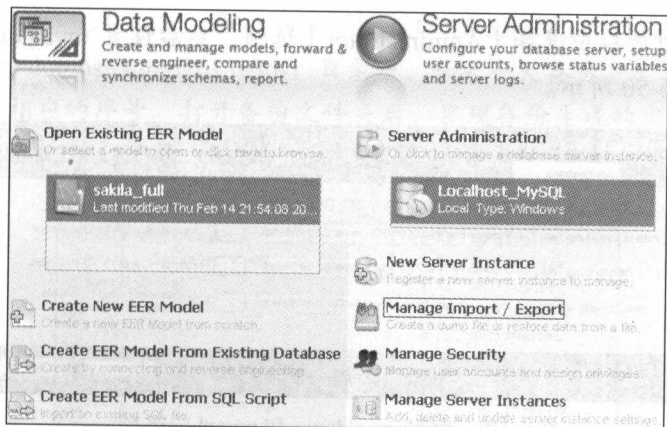


图 6-47 单击【Manage Import/Export】按钮

步骤 02 在弹出的窗口中，选择左侧的【DATA EXPORT / RESTORE】选项，该选项下面有两个选项，其中【Data Export】选项表示数据库导出，【Data Import/Restore】选项表示数据库还原选项，如图 6-48 所示。

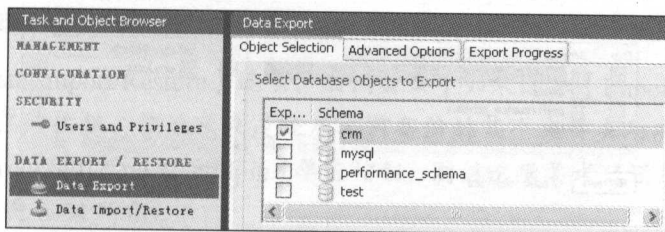


图 6-48 选择【Data Import/Restore】选项

步骤 03 在开始备份之前，用户还需要设置 MySQL Workbench 中备份 MySQL 的 mysqldump 的路径，MySQL Workbench 默认的 mysqldump 版本是 5.5 的，如果需要备份更好版本的数据库的话，可能会发生错误，下面先设置下 mysqldump 的版本。选择【Edit】|【Preferences】菜单命令，如图 6-49 所示。

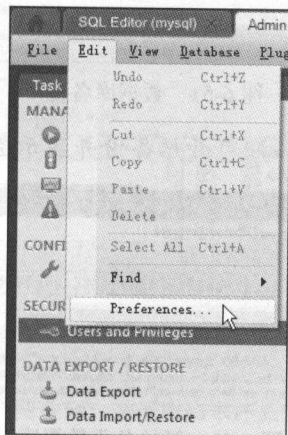


图 6-49 选择【Edit】|【Preferences】菜单命令

步骤 04 在弹出的对话框中选择【Administrator】选项，然后设置 mysqldump 和 mysql 的路径。如图 6-50 所示。

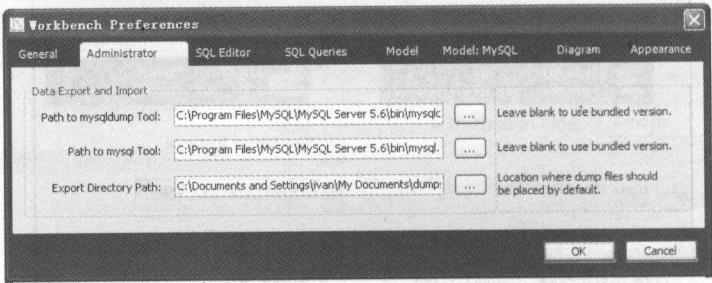


图 6-50 设置 mysqldump 和 mysql 的路径

步骤 05 单击【OK】按钮，返回到数据库备份的页面，选择需要备份的数据库，在 Options 选项中，可以选择【Export to Dump Project Folder】选项，此时备份数据库数据会将数据备份到系统默认的路径中，如图 6-51 所示。

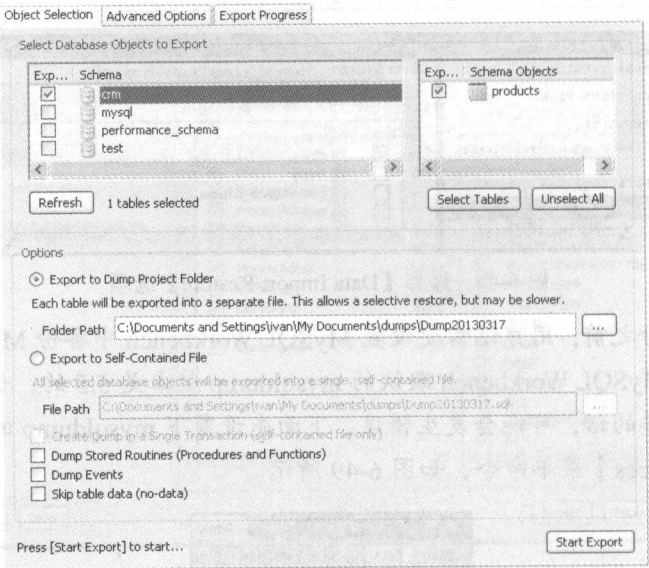


图 6-51 数据库备份

步骤 06 单击【Start Export】按钮，即可开始备份并显示备份的进度。如图 6-52 所示。

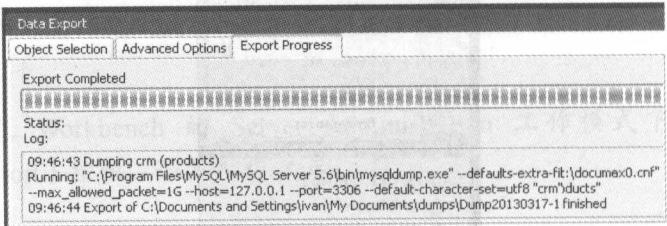


图 6-52 开始备份数据库

步骤 07 数据库数据备份到 C:\Documents and Settings\ivan\My Documents\dumps\Dump2013036-1 目录中, 如果一天备份多次, 那么文件夹日期后面会加上相应的序号。打开备份文件夹后, 发现备份生成的文件都是以“数据库名_表名”的格式来备份脚本文件的, 如图 6-53 所示。

| 名称 | 大小 | 类型 |
|------------------------------|--------|------------------|
| crm_products.sql | 3 KB | DbVisualizer ... |
| mysql_columns_priv.sql | 3 KB | DbVisualizer ... |
| mysql_db.sql | 4 KB | DbVisualizer ... |
| mysql_event.sql | 4 KB | DbVisualizer ... |
| mysql_func.sql | 3 KB | DbVisualizer ... |
| mysql_help_category.sql | 4 KB | DbVisualizer ... |
| mysql_help_keyword.sql | 10 KB | DbVisualizer ... |
| mysql_help_relation.sql | 12 KB | DbVisualizer ... |
| mysql_help_topic.sql | 492 KB | DbVisualizer ... |
| mysql_innodb_index_stats.sql | 3 KB | DbVisualizer ... |
| mysql_innodb_table_stats.sql | 3 KB | DbVisualizer ... |

图 6-53 备份的数据库脚本文件

6.4.3 还原 MySQL 数据库

数据库备份完成后, 如果需要可以进行还原数据库的操作。具体操作步骤如下。

步骤 01 单击【Data Import/Restore】选项, 右侧窗口中如果选择【Import from Dump Project Folder】单选按钮, 该选项表是还原之前备份的某个文件夹所有的脚本; 如果选择【Import from Self-Contained File】单选按钮, 则该选项要求只还原其中的某个备份的脚本。如图 6-54 所示。

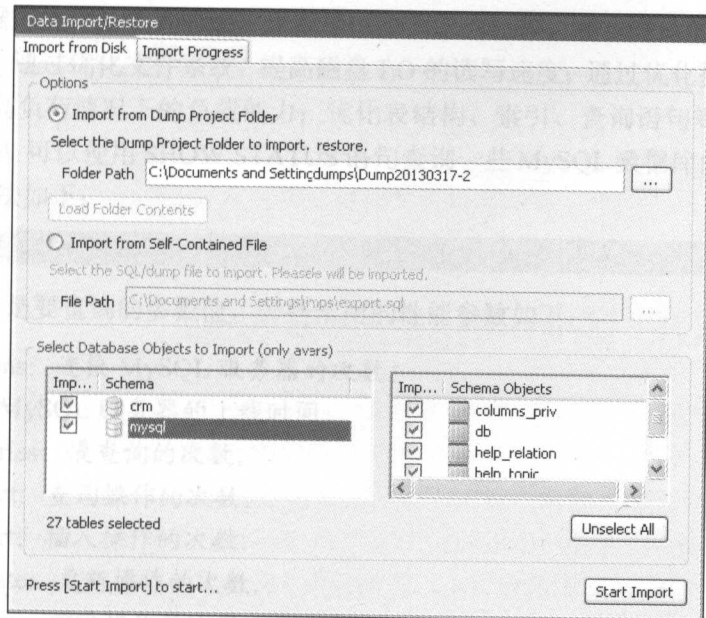


图 6-54 数据库脚本还原

步骤 02 本实例选择【 Import from Dump Project Folder 】选项，然后选择需要还原的数据库和表，单击【 Start Import 】按钮。系统将自动进行还原操作，如图 6-55 所示。

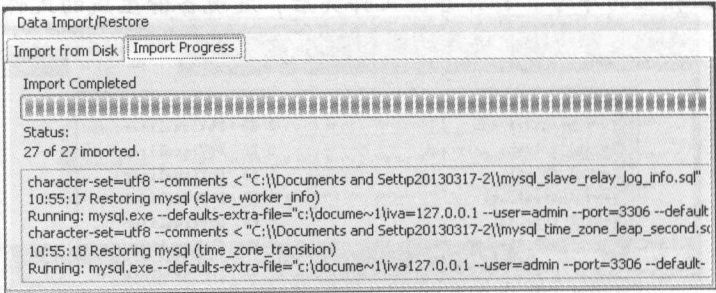


图 6-55 数据库还原的进度条

6.5 小结

MySQL Workbench 是 MySQL 数据库中非常好用的管理工具之一，MySQL Workbench 5.2 提供了图形化界面下的数据库基本的管理，数据库建立物理模型，以及通过物理模型转换成执行的 SQL 脚本。另外 MySQL Workbench 5.2 提供了对 MySQL 数据库性能的监控，用户的管理，以及备份和还原数据库数据等功能，非常方便 MySQL 数据库开发人员和管理人员。

第 7 章

◀ SQL性能优化 ▶

对于应用而言，初期数据量不是很大的情况下，SQL 还不会很明显地制约应用的效率，随着应用开始使用，数据开始持续增长，很多 SQL 语句开始逐渐显露出性能问题，对于海量数据存储的应用而言，SQL 执行的效率更加制约了应用的瓶颈，所以优化 SQL 很有必要性，本章主要介绍了如何通过 SQL 来优化提升数据库的性能。

7.1 优化简介

优化 MySQL 数据库是数据库管理员和数据库开发人员的必备技能。MySQL 优化，一方面是找出系统的瓶颈，提高 MySQL 数据库整体的性能；另一方面需要合理的结构设计和参数调整，以提高用户操作响应的速度；同时还要尽可能节省系统资源，以便系统可以提供更大负荷的服务。本节将为读者介绍优化的基本知识。

MySQL 数据库优化是多方面的，原则是减少系统的瓶颈，减少资源的占用，增加系统的反应速度。例如，通过优化文件系统，提高磁盘 I/O 的读写速度；通过优化操作系统调度策略，提高 MySQL 在高负荷情况下的负载能力；优化表结构、索引、查询语句等使查询响应更快。

在 MySQL 中，可以使用 SHOW STATUS 语句查询一些 MySQL 数据库的性能参数。SHOW STATUS 语句语法如下：

```
SHOW STATUS LIKE 'value';
```

其中，value 是要查询的参数值，一些常用的性能参数如下：

- Connections: 连接 MySQL 服务器的次数。
- Uptime: MySQL 服务器的上线时间。
- Slow_queries: 慢查询的次数。
- Com_select: 查询操作的次数。
- Com_insert: 插入操作的次数。
- Com_update: 更新操作的次数。
- Com_delete: 删除操作的次数。

如果查询 MySQL 服务器的连接次数，可以执行如下语句：

```
SHOW STATUS LIKE 'Connections';
```

如果查询 MySQL 服务器的慢查询次数，可以执行如下语句：

```
SHOW STATUS LIKE 'Slow_queries';
```

查询其他参数的方法和上述两个参数的查询方法相同。慢查询次数参数可以结合慢查询日志，找出慢查询语句，然后针对慢查询语句进行表结构优化或者查询语句优化。

7.2 MySQL Query Optimizer 概述

MySQL 中有一个专门负责优化 SELECT 语句的优化器模块，该优化器模块就是 Query Optimizer，它的主要功能是将客户端传递的查询语句，结合数据库系统收集的各种统计信息，为客户端的 SQL 进行查询优化，最终给出优化之后的查询语句。

当用户模块发送请求给 MySQL 命令解析器之后，MySQL 命令解析器会将用户查询语句发送给 MySQL 查询优化器（Query Optimizer），然后对查询语句进行优化。例如常量转化，无效内容删除，常量计算等待等等。Query Optimizer 会分析出最优化数据检索的方式，也就是常说的执行计划。

MySQL Query Optimizer 的所有工作都是基于 Query Tree 的。Query Tree 是通过优化实现 DBXP 的经典数据结构和 Tree 构造器而生成的一个指导完成一个 Query 语句的请求所需要处理的工作步骤，通过 Query Tree 可以清楚地了解到一个 Query 完成所需要经历的步骤，每一步的数据来源，以及处理方式等信息。

MySQL Query Optimizer 是在 CBO（Cost Base Optimizer）的基础上增加了一个 Heuristic Optimize（启发式优化）的功能，MySQL Query Optimizer 在优化一个查询语句并选择出最优执行计划的时候增加了某些特定的规则。

7.3 SQL 语句优化的基本思路

在进行 SQL 语句优化的时候，首先要做的是找到问题的所在，然后才能有针对性地解决问题。

首先从业务上理解客户的真正需求，除了通过很复杂的 SQL 查询外，是否有更为简洁的方法得到客户的需求，这是进行优化 SQL 之前优先要考虑的。另外，对 SQL 进行优化时需要注意，SQL 查询返回的结果集是否太大，数据量太大对客户来讲意义不大，如果返回的结果集太大往往需要消耗额外的磁盘读写，除非客户真正需要这么多数据，否则可以调整查询条件，从而降低查询结果集的大小。

在分析如何优化 SQL 语句之前，需要了解优化查询语句的基本思路。通常，查询语句优化思路主要体现在以下几个方面：

1. 优化更需要优化的查询

什么样的查询是更需要优化的查询呢？这个要从整个系统的影响来考虑，什么样的优化能给系统整体带来更大的收益，就更需要优化的查询。例如高并发低消耗的查询对整个系统的影响远比低并发高消耗的查询大。下面通过一个例子来理解。

假设有一个查询每分钟执行 10000 次，每次需要 30 个读写操作，另外一个查询每分钟执行 100 次，每次需要 3000 个读写操作。从读写消耗方面分析，两个查询每分钟所消耗的读写操作总数是一样。如果优化第一个查询，从 30 个读写降低到 25 个读写，则每分钟总的读写操作为 2250000 个。如果优化第二个查询，想达到第一个优化效果，则需要降低 500 读写操作。可见第一个优化方案比第二个优化方案容易的多。

另外，比较 CPU 方面的消耗，原理和上面的完全一样。只要让第一个查询稍微节省一小块资源，就可以让整个系统节省出一大块资源，尤其是在排序、分组这些对 CPU 消耗比较多的操作中非常突出。

对整个系统的影响来分析。一个频繁执行的高并发查询的危险性比一个低并发的查询要大很多。当一个低并发的查询走错执行计划，所带来的影响主要只是该查询的请求者的体验会变差，对整体系统的影响并不会特别的突出，至少还属于可控范围。如果一个高并发的查询走错了执行计划，那所带来的后果很可能就是灾难性的。

2. 定位优化对象的性能瓶颈

优化查询之前，首先要看该优化对象的性能瓶颈是读写问题还是 CPU 耗损问题，到底是数据运算方面耗费了太多资源，还是在数据访问上耗费了太多的时间等。

在 MySQL 5.0 系列版本中，可以通过系统自带的 PROFILING 功能很清楚地找出一个查询的瓶颈所在。而在早期的版本中，只能通过自行分析查询的各个操作步骤，找到性能耗损的瓶颈所在。

3. 明确的优化目标

当找到优化的性能瓶颈之后，就需要制订明确的优化目标。首先需要搞清楚数据库目前的整体状态，这样才能知道数据库所能承受的最大压力。然后是清楚地知道数据库中与该查询相关的数据库对象的各种信息，这样才能知道该查询在最理想情况下需要消耗多少资源，最糟糕又需要消耗多少资源。最后是了解该查询在整个应用系统中所实现的功能，这样才能知道该查询所实现的功能点在整个应用系统中的重要地位，可以大概地分析出该查询可以占用的系统资源比例，也能够知道该查询的效率给客户带来的体验影响到底有多大。

明确了上述信息以后，就可以得出该查询应该满足的性能范围，这也就是优化目标范围，然后通过寻找相应的优化手段来解决问题了。如果该查询实现的应用系统功能比较重要，

那么就需要进一步优化，可以在其他某些方面作出一些让步与牺牲，比如调整设计，调整索引组成等。

4. 充分使用 EXPLAIN 和 PROFILE 工具对 SQL 语句进行分析

有了优化目标以后，就可以动手优化操作了。首先从 Explain 工具下手，通过此工具，可以查看该查询的执行计划。在优化之前，必须要有一个预订的执行计划，然后借助 Explain 工具来验证调整后的结果是否满足预订的执行计划。对于不符合预期的执行计划需要不断分析查询的写法和数据库对象的信息，继续调整尝试，直至得到预期的结果。在不断调整测试的过程中，如果发现 MySQL Optimizer 所选择的执行计划的实际执行效果确实比自己预设的要好，还是应该选择使用 MySQL optimizer 所生成的执行计划。

7.4 利用 EXPLAIN 分析查询语句

通过对查询语句的分析，可以了解查询语句的执行情况，找出查询语句执行的瓶颈，从而优化查询语句。MySQL 中提供了 EXPLAIN 语句，用来分析查询语句。本小节将为读者介绍使用 EXPLAIN 语句分析查询语句的方法。

7.4.1 EXPLAIN 语句的基本语法

EXPLAIN 语句的基本语法如下：

```
EXPLAIN [EXTENDED] SELECT select_options
```

使用 EXTENDED 关键字，EXPLAIN 语句将产生附加信息。select_options 是 SELECT 语句的查询选项，包括 FROM WHERE 子句等。

执行该语句，可以分析 EXPLAIN 后面的 SELECT 语句的执行情况，并且能够分析出所查询的表的一些特征。

使用 EXPLAIN 语句来分析 1 个查询语句，执行如下语句：

```
mysql> EXPLAIN SELECT * FROM fruits;
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | fruits | ALL | NULL | NULL | NULL | NULL | 16 |  |
+----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

下面对查询结果进行解释。

1. id

SELECT 识别符。这是 SELECT 的查询序列号。id 的值越大优先级别越高，越先被执行。如果 id 相同，执行顺序由上至下。

2. select_type

select_type 表示 SELECT 语句的类型。它可以是以下几种取值：

(1) SIMPLE

SIMPLE 表示简单查询。其中不包括连接查询和子查询。例如下面的例子：

```
mysql> explain select * from books where id = 2\G;
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: books
        type: const
possible_keys: PRIMARY
         key: PRIMARY
        key_len: 4
         ref: const
        rows: 1
       Extra:
1 row in set (0.00 sec)
```

(2) PRIMARY

PRIMARY 表示主查询或者是最外层的查询语句。例如下面的例子：

```
mysql> explain select *from (select * from books where id = 23 )a \G;
***** 1. row *****
      id: 1
  select_type: PRIMARY
        table: <derived2>
        type: system
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
        rows: 1
       Extra:
***** 2. row *****
      id: 2
  select_type: DERIVED
        table: books
        type: const
```



```
possible_keys: PRIMARY
  key: PRIMARY
  key_len: 4
  ref:
  rows: 1
  Extra:
2 rows in set (0.00 sec)
```

(3) UNION 和 UNION RESULT

UNION 表示连接查询的第 2 个或后面的查询语句，不依赖于外部查询的结果集。而 UNION RESULT 表示 UNION 查询的结果集。例如下面的例子：

```
mysql> explain select * from books where id =2 union all select * from books
-> where id = 3 \G;
***** 1. row *****
      id: 1
  select_type: PRIMARY
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
      key_len: 4
      ref: const
      rows: 1
      Extra:
***** 2. row *****
      id: 2
  select_type: UNION
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
      key_len: 4
      ref: const
      rows: 1
      Extra:
***** 3. row *****
      id: NULL
  select_type: UNION RESULT
      table: <union1,2>
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: NULL
```

```
Extra:
3 rows in set (0.08 sec)
```

(4) DEPENDENT UNION

DEPENDENT UNION 连接查询中的第 2 个或后面的 SELECT 语句，取决于外面的查询。
例如下面的例子：

```
mysql> explain select * from books where id in (select id from books where id
=
-> 23 union all select id from books where id = 3) \G
***** 1. row *****
      id: 1
select_type: PRIMARY
      table: books
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 10088
Extra: Using where
***** 2. row *****
      id: 2
select_type: DEPENDENT SUBQUERY
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
      key_len: 4
      ref: const
      rows: 1
Extra: Using index
***** 3. row *****
      id: 3
select_type: DEPENDENT UNION
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
      key_len: 4
      ref: const
      rows: 1
Extra: Using index
***** 4. row *****
      id: NULL
select_type: UNION RESULT
```



```

table: <union2,3>
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: NULL
Extra:
4 rows in set (0.00 sec)

```

(5) SUBQUERY

SUBQUERY 子查询中的第 1 个 SELECT 语句，不依赖于外部查询的结果集。例如下面的例子：

```

mysql> explain select * from books where id = (select id from books where id=3)
\G;
***** 1. row *****
      id: 1
select_type: PRIMARY
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
     key_len: 4
       ref: const
      rows: 1
     Extra:
***** 2. row *****
      id: 2
select_type: SUBQUERY
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
     key_len: 4
       ref:
      rows: 1
     Extra: Using index
2 rows in set (0.00 sec)

```

(6) DEPENDENT SUBQUERY

DEPENDENT SUBQUERY 子查询中的第 1 个 SELECT，取决于外面的查询。例如下面的例子：

```

mysql> explain select id from books where id in (select id from books where id=3)
\G;

```



```

***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: books
        type: index
possible_keys: NULL
      key: PRIMARY
     key_len: 4
        ref: NULL
       rows: 10088
    Extra: Using where; Using index
***** 2. row *****
      id: 2
    select_type: DEPENDENT SUBQUERY
      table: books
        type: const
possible_keys: PRIMARY
      key: PRIMARY
     key_len: 4
        ref: const
       rows: 1
    Extra: Using index
2 rows in set (0.01 sec)

```

(7) DERIVED

DERIVED 查询类型用于 from 子句里有子查询的情况。MySQL 会递归执行这些子查询，把结果放在临时表里。例如下面的例子：

```

mysql> explain select * from (select * from books where id = 23) a \G;
***** 1. row *****
      id: 1
    select_type: PRIMARY
      table: <derived2>
        type: system
possible_keys: NULL
      key: NULL
     key_len: NULL
        ref: NULL
       rows: 1
    Extra:
***** 2. row *****
      id: 2
    select_type: DERIVED
      table: books
        type: const
possible_keys: PRIMARY

```

```
key: PRIMARY
key_len: 4
ref:
rows: 1
Extra:
2 rows in set (0.00 sec)
```

3. table

table 表示显示这一行的数据是关于哪张表的。有可能看到的不是真实的表名字，而是 derivedx (X 是个数字)。例如下面的例子：

```
mysql> explain select * from (select * from (select *from books where id=23)
a)b \G;
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table      | type   | possible_keys | key      | key_len |
ref | rows | Extra |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | PRIMARY | <derived2> | system | NULL | NULL | NULL | NULL | 1 |
| 2 | DERIVED | <derived3> | system | NULL | NULL | NULL | NULL | 1 |
| 3 | DERIVED | books | const | PRIMARY | PRIMARY | 4 | 1 |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

4. type

type 表示表的连接类型。下面按照从最佳类型到最差类型的顺序给出各种连接类型：

(1) system

该表是仅有一行的系统表，这是 const 连接类型的一个特例。例如下面的例子：

```
mysql> explain select * from (select * from (select *from books where id=23)
a)b
\G;
***** 1. row *****
id: 1
select_type: PRIMARY
table: <derived2>
type: system
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 1
Extra:
```

```

***** 2. row *****
      id: 2
    select_type: DERIVED
      table: <derived3>
      type: system
possible_keys: NULL
      key: NULL
     key_len: NULL
      ref: NULL
      rows: 1
    Extra:
***** 3. row *****
      id: 3
    select_type: DERIVED
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
     key_len: 4
      ref:
      rows: 1
    Extra:
3 rows in set (0.00 sec)

```

(2) const

数据表最多只有一个匹配行，它将在查询开始时被读取，并在余下的查询优化中作为常量对待。const 表查询速度很快，因为它们只读取一次。const 用于使用常数值比较 PRIMARY KEY 或 UNIQUE 索引的所有部分的场合。例如下面的例子：

```

mysql> explain select * from books where id = 2\G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: books
      type: const
possible_keys: PRIMARY
      key: PRIMARY
     key_len: 4
      ref: const
      rows: 1
    Extra:
1 row in set (0.00 sec)

```

(3) eq_ref

对于每个来自前面的表的行组合，从该表中读取一行。当一个索引的所有部分都在查询中

使用并且索引是 UNIQUE 或 PRIMARY KEY 时,即可使用这种类型。

eq_ref 可以用于使用 “=” 操作符比较带索引的列。比较值可以为常量或一个在该表前面所读取的表的列的表达式。例如下面的例子:

```
mysql> explain select * from t1,t2 where t1.id = t2.id \G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: t2
       type: ALL
possible_keys: NULL
       key: NULL
      key_len: NULL
       ref: NULL
      rows: 3122
    Extra:
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: t1
       type: ref
possible_keys: id_t1_id
       key: id_t1_id
      key_len: 5
       ref: test.t2.id
      rows: 1
    Extra: Using where
2 rows in set (0.05 sec) 2 rows in set (0.00 sec)
```

(4) ref

对于来自前面的表的任意行组合,将从该表中读取所有匹配的行。这种类型用于索引既不是 UNIQUE 也不是 PRIMARY KEY 的情况,或者查询中使用了索引列的左子集,即索引中左边的部分列组合。ref 可以用于使用 = 或 <=> 操作符的带索引的列。例如下面的例子:

```
mysql> create index id_t1_id on t1(id);
Query OK, 3072 rows affected (0.17 sec)
Records: 3072 Duplicates: 0 Warnings: 0

mysql> explain select * from t1,t2 where t1.id = t2.id \G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: t2
       type: ALL
possible_keys: NULL
```

```

key: NULL
key_len: NULL
ref: NULL
rows: 3122
Extra:
***** 2. row *****
    id: 1
select_type: SIMPLE
  table: t1
   type: ref
possible_keys: id_t1_id
   key: id_t1_id
  key_len: 5
   ref: test.t2.id
   rows: 1
  Extra: Using where
2 rows in set (0.00 sec)

```

(5) ref_or_null

该连接类型如同 ref，但是添加了 MySQL 可以专门搜索包含 NULL 值的行。在解决子查询中经常使用该连接类型的优化。例如下面的例子：

```

mysql> explain select * from t1 where id =37 or id is null \G;
***** 1. row *****
    id: 1
select_type: SIMPLE
  table: t1
   type: ref_or_null
possible_keys: id_t1_id
   key: id_t1_id
  key_len: 5
   ref: const
   rows: 2
  Extra: Using where
1 row in set (0.00 sec)

```

(6) index_merge

该连接类型表示使用了索引合并优化方法。在这种情况下，key 列包含了使用的索引的清单，key_len 包含了使用的索引的最长的关键元素。

(7) unique_subquery

该类型替换了下面形式的 IN 子查询的 ref:

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

unique_subquery 是一个索引查找函数，可以完全替换子查询，效率更高。

下面的例子中，id 为表 books 中的 primary key。

```
mysql> explain select * from books where id in (select id from books where id<20
) \G;
***** 1. row *****
      id: 1
  select_type: PRIMARY
        table: books
         type: ALL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
         rows: 10088
      Extra: Using where
***** 2. row *****
      id: 2
  select_type: DEPENDENT SUBQUERY
        table: books
         type: unique_subquery
possible_keys: PRIMARY
         key: PRIMARY
        key_len: 4
         ref: func
         rows: 1
      Extra: Using index; Using where
2 rows in set (0.00 sec)
```

(8) index_subquery

该连接类型类似于 unique_subquery，可以替换 IN 子查询，但只适合下列形式的子查询中的非唯一索引。例如下面的例子：

```
mysql> explain select * from t1 where id in (select id from t1 where id<20 ) \G;
***** 1. row *****
      id: 1
  select_type: PRIMARY
        table: t1
         type: ALL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
         rows: 2972
      Extra: Using where
***** 2. row *****
      id: 2
```



```

select_type: DEPENDENT SUBQUERY
table: t1
type: index_subquery
possible_keys: id_t1_id
key: id_t1_id
key_len: 5
ref: func
rows: 1
Extra: Using index; Using where
2 rows in set (0.00 sec)

```

(9) range

只检索给定范围的行，使用一个索引来选择行。**key** 列显示使用了哪个索引。**key_len** 包含所使用索引的最长关键元素。

当使用=、<、>、>=、<=、IS NULL、<=>、BETWEEN 或者 IN 操作符，用常量比较关键字列时，类型为 **range**。

下面介绍几种检索指定行的情况：

```

SELECT * FROM tbl_name
WHERE key_column = 10;

SELECT * FROM tbl_name
WHERE key_column BETWEEN 10 and 20;
SELECT * FROM tbl_name
WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
WHERE key_part1= 10 AND key_part2 IN (10,20,30);

```

例如具体的例子：

```

mysql> explain select * from t1 where id = 32 or id = 43 \G;
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: t1
      type: range
possible_keys: id_t1_id
       key: id_t1_id
    key_len: 5
       ref: NULL
      rows: 2
    Extra: Using where
1 row in set (0.00 sec)

```

(10) index

该连接类型与 ALL 相同，除了只扫描索引树。这通常比 ALL 快，因为索引文件通常比数据文件小。

(11) ALL

对于前面的表的任意行组合，进行完整的表扫描。如果表是第一个没标记 `const` 的表，这样不好，并且在其他情况下很差。通常可以增加更多的索引来避免使用 ALL 连接。

5. possible_keys

`possible_keys` 列指出 MySQL 能使用哪个索引在该表中找到行。如果该列是 NULL，则没有相关的索引。在这种情况下，可以通过检查 `WHERE` 子句看它是否引用某些列或适合索引的列来提高查询性能。如果是这样，可以创建适合的索引来提高查询的性能。

6. key

`key` 表示查询实际使用到的索引，如果没有选择索引，该列的值是 NULL。要想强制 MySQL 使用或忽视 `possible_keys` 列中的索引，在查询中使用 `FORCE INDEX`、`USE INDEX` 或者 `IGNORE INDEX`。参见 `SELECT` 语法。

7. key_len

`key_len` 表示 MySQL 选择的索引字段按字节计算的长度，如果键是 NULL，则长度为 NULL。注意通过 `key_len` 值可以确定 MySQL 将实际使用一个多列索引中的几个字段。

8. ref

`ref` 表示使用哪个列或常数与索引一起来查询记录。

9. rows

`rows` 显示 MySQL 在表中进行查询时必须检查的行数。

10. extra

`extra` 列显示 MySQL 在处理查询时的详细信息，主要包括如下：

- `using index`: 出现这个说明 MySQL 使用了覆盖索引，避免访问了表的数据行，效率不错。
- `using where`: 这说明服务器在存储引擎收到行后将进行过滤。
- `using temporary`: 这意味着 MySQL 对查询结果进行排序的时候使用了一张临时表。
- `using filesort`: 这个说明 MySQL 会对数据使用一个外部的索引排序。

当出现 `using temporary` 和 `using filesort` 时，需要对查询语句进行优化操作。

7.4.2 EXPLAIN 语句分析实例

掌握了 EXPLAIN 用法之后，我们可以通过分析 EXPLAIN 返回的每一项结果，就能清楚地

知道查询大致的运行时间。如果查询没有用到索引、或者扫描的行过多，那么需要改变查询的方式或者建立索引。

下面通过一个案例来进一步学习 EXPLAIN 的用法，具体操作步骤如下：

步骤 01 创建数据表 emp，命令如下：

```
mysql> create table emp(
-> id integer primary key auto_increment,
-> deptno integer,
-> col3 integer,
-> col4 integer
-> );
Query OK, 0 rows affected (0.06 sec)
```

步骤 02 插入若干条记录，命令如下：

```
mysql> insert into emp values(1,1,2,3);
Query OK, 1 row affected (0.00 sec)

mysql> insert into emp values(2,3,4,5);
Query OK, 1 row affected (0.00 sec)

mysql> insert into emp values(3,4,5,6);
Query OK, 1 row affected (0.00 sec)
```

步骤 03 查询 deptno 为 2 并且 col3 大于 1，col4 最大的 id，分析上述查询语句，命令如下：

```
mysql> explain select id from emp where deptno=2 and col3>1 order by col4 desc
1
imit 1 \G;
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: emp
         type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
         ref: NULL
        rows: 6
      Extra: Using where; Using filesort
1 row in set (0.00 sec)
```

步骤 04 此时，type 的值是 ALL 类型（最差的一种连接类型），extra 中出现了 Using filesort，此时需要进行优化，首先建立一个联合索引进行最简单的优化，命令如下。

```
mysql> alter table emp add index index_1(deptno,col3,col4);
```



```
Query OK, 6 rows affected (0.06 sec)
Records: 6 Duplicates: 0 Warnings: 0
```

步骤 05 再使用 EXPLAIN 分析查询语句的情况，命令如下：

```
mysql> explain select id from emp where deptno=2 and col3>1 order by col4 desc
1
imit 1 \G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
        type: ref
possible_keys: index_1
         key: index_1
        key_len: 5
         ref: const
         rows: 3
      Extra: Using where; Using index; Using filesort
1 row in set (0.05 sec)
```

显然，type 的值变成了 ref，但是 extra 里面仍然是 Using filesort，建立了索引，可是为什么没起到作用呢？

这是因为按照 BTree 索引的工作原理，先排序 deptno，如果遇到相同的 deptno 就会进行对 col3 进行排序，如果遇到相同的 col3 就会在对 col4 信息排序。因为 col3>1，则索引无法对后面的 col4 列进行排序，下面进一步进行优化，命令如下：

```
mysql> drop index index_1 on emp;
Query OK, 6 rows affected (0.02 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> alter table emp add index index_1(deptno,col4);
Query OK, 6 rows affected (0.01 sec)
Records: 6 Duplicates: 0 Warnings: 0

mysql> explain select id from emp where deptno=2 and col3>1 order by col4 desc
1
imit 1 \G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
        type: ref
possible_keys: index_1
         key: index_1
        key_len: 5
```

```

      ref: const
      rows: 2
      Extra: Using where
1 row in set (0.00 sec)

```

从结果可以看出，优化后的效果非常不错。

下面接着讲述一个多表查询优化的例子，具体操作步骤如下：

步骤 01 创建两个表，并插入数据，命令如下：

```

mysql> create table table1(
-> id integer primary key auto_increment,
-> tid integer not null
-> );
Query OK, 0 rows affected (0.00 sec)

```

```

mysql> create table table2(
-> id integer primary key auto_increment,
-> tid integer not null
-> );
Query OK, 0 rows affected (0.00 sec)

```

步骤 02 分析多表查询语句，命令如下：

```

mysql> explain select * from table1 left join table2 on table1.tid = table2.tid
\G;
***** 1. row *****
      id: 1
select_type: SIMPLE
      table: table1
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 4019
      Extra:
***** 2. row *****
      id: 1
select_type: SIMPLE
      table: table2
      type: ALL
possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 4072

```

```
Extra:
2 rows in set (0.00 sec)
```

步骤 03 创建索引进行优化, 命令如下:

```
mysql> alter table table2 add index(tid);
Query OK, 4096 rows affected (0.20 sec)
Records: 4096 Duplicates: 0 Warnings: 0
```

步骤 04 重新分析多表查询语句, 命令如下:

```
mysql> explain select * from table1 left join table2 on table1.tid = table2.tid
\G;
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: table1
         type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
         ref: NULL
        rows: 4019
      Extra:
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: table2
         type: ref
possible_keys: tid
         key: tid
      key_len: 4
         ref: test.table1.tid
        rows: 1
      Extra: Using index
2 rows in set (0.00 sec)
```

对于 table2 的连接类型此时发生了改变, 变成了 ref, 同时 rows 有了很大的提升, 从 4019*4072 提升到 4019*1, 优化的效果很明显。

7.5 利用 Profiling 分析查询语句

MySQL 的 Profiler 是一个使用非常方便的查询诊断分析工具, 通过该工具可以获取一条查询在整个执行过程中多种资源的消耗情况, 例如内存消耗、I/O 消耗和 CPU 消耗等。

Profile 的语法规则如下:

```
SHOW PROFILE [type [,type] ...]
[FOR QUERY n]
[LIMIT row_count [OFFSET offset]]
```

其中 type 参数的可选项含义如下。

- ALL: 显示所有的信息。
- BLOCK IO: 显示输入输出操作阻塞的数量。
- CONTEXT SWITCHES: 显示自动或非自动 CONTEXT SWITCHES 的数量。
- CPU: 显示系统和用户 CPU 使用的时间。
- IPC: 显示信息发送和接收的数量。
- MEMORY: 内存的信息。
- PAGE FAULTS: 显示主要的 PAGE FAULTS 数量。
- SOURCE: 显示函数的名称, 并且显示函数所在文件的名称和行数。
- SWAPS: 显示 SWAP 数量。

下面举例介绍 MySQL Profiler 的使用方法。

步骤 01 查询 profiling 是否开启, 命令如下:

```
mysql> select @@profiling;
+-----+
| @@profiling |
+-----+
|          0 |
+-----+
1 row in set (0.02 sec)
```

步骤 02 开启 profiling, 命令如下:

```
mysql> SET profiling = 1;
Query OK, 0 rows affected (0.02 sec)

mysql> select @@profiling;
+-----+
| @@profiling |
+-----+
|          1 |
+-----+
1 row in set (0.00 sec)
```

步骤 03 执行若干条 SQL 查询语句, 命令如下:

```
mysql> use mysql;
mysql> SELECT DATABASE();
```

```
mysql> select * from user limit 1;
mysql> select count(*) from user group by sexal;
mysql> SELECT DATABASE();
mysql> show databases;
mysql> select count(*) from books;
```

步骤 04 查看上述语句的 Profiling 情况, 命令如下:

```
mysql> show profiles;
+-----+-----+-----+
| Query_ID | Duration | Query |
+-----+-----+-----+
| 1 | 0.00020775 | select @@profiling |
| 2 | 0.02026325 | use mysql |
| 3 | 0.01109650 | SELECT DATABASE() |
| 4 | 0.06095950 | select * from user limit 1 |
| 5 | 0.01308450 | select count(*) from user group by sexal |
| 6 | 0.00016825 | SELECT DATABASE() |
| 7 | 0.06119525 | show databases |
| 8 | 0.08667300 | select count(*) from books |
+-----+-----+-----+
8 rows in set (0.00 sec)
```

步骤 05 查看 Query_ID=8 语句的 profiles, 命令如下:

```
mysql> show profile for query 8;
+-----+-----+
| Status | Duration |
+-----+-----+
| starting | 0.000085 |
| Opening tables | 0.000019 |
| System lock | 0.000005 |
| Table lock | 0.000009 |
| init | 0.000015 |
| optimizing | 0.000006 |
| statistics | 0.000012 |
| preparing | 0.000009 |
| executing | 0.000005 |
| Sending data | 0.000205 |
| end | 0.000005 |
| end | 0.000003 |
| query end | 0.000005 |
| freeing items | 0.000043 |
| closing tables | 0.000006 |
| logging slow query | 0.000003 |
| cleaning up | 0.000004 |
+-----+-----+
```

17 rows in set (0.00 sec)

步骤 06 查看 Query_ID=8 语句的 CPU 消耗情况, 命令如下:

```
mysql> show profile cpu for query 8;
```

| Status | Duration | CPU_user | CPU_system |
|--------------------|----------|----------|------------|
| starting | 0.000085 | NULL | NULL |
| Opening tables | 0.000019 | NULL | NULL |
| System lock | 0.000005 | NULL | NULL |
| Table lock | 0.000009 | NULL | NULL |
| init | 0.000015 | NULL | NULL |
| optimizing | 0.000006 | NULL | NULL |
| statistics | 0.000012 | NULL | NULL |
| preparing | 0.000009 | NULL | NULL |
| executing | 0.000005 | NULL | NULL |
| Sending data | 0.000205 | NULL | NULL |
| end | 0.000005 | NULL | NULL |
| end | 0.000003 | NULL | NULL |
| query end | 0.000005 | NULL | NULL |
| freeing items | 0.000043 | NULL | NULL |
| closing tables | 0.000006 | NULL | NULL |
| logging slow query | 0.000003 | NULL | NULL |
| cleaning up | 0.000004 | NULL | NULL |

17 rows in set (0.00 sec)

步骤 07 查看 Query_ID=8 语句的 I/O 消耗情况, 命令如下:

```
mysql> show profile block io for query 8;
```

| Status | Duration | Block_ops_in | Block_ops_out |
|----------------|----------|--------------|---------------|
| starting | 0.000085 | NULL | NULL |
| Opening tables | 0.000019 | NULL | NULL |
| System lock | 0.000005 | NULL | NULL |
| Table lock | 0.000009 | NULL | NULL |
| init | 0.000015 | NULL | NULL |
| optimizing | 0.000006 | NULL | NULL |
| statistics | 0.000012 | NULL | NULL |
| preparing | 0.000009 | NULL | NULL |
| executing | 0.000005 | NULL | NULL |
| Sending data | 0.000205 | NULL | NULL |
| end | 0.000005 | NULL | NULL |
| end | 0.000003 | NULL | NULL |
| query end | 0.000005 | NULL | NULL |


```
| freeing items      | 0.000043 |      NULL |      NULL |
| closing tables    | 0.000006 |      NULL |      NULL |
| logging slow query | 0.000003 |      NULL |      NULL |
| cleaning up       | 0.000004 |      NULL |      NULL |
+-----+-----+-----+-----+
17 rows in set (0.00 sec)
```

7.6 合理地使用索引

索引是数据库优化中最重要的方法之一，本章节主要讲解如何合理地使用索引。

7.6.1 索引对查询速度的影响

MySQL 中提高性能的一个最有效的方式就是对数据表设计合理的索引。索引提供了高效访问数据的方法，并且加快查询的速度，因此，索引对查询的速度有着至关重要的影响。使用索引可以快速地定位表中的某条记录，从而提高数据库查询的速度，提高数据库的性能。本小节将为读者介绍索引对查询速度的影响。

如果查询时没有使用索引，查询语句将扫描表中的所有记录。在数据量大的情况下，这样查询的速度会很慢。如果使用索引进行查询，查询语句可以根据索引快速定位到待查询记录，从而减少查询的记录数，达到提高查询速度的目的。

下面是查询语句中不使用索引和使用索引的对比。首先，分析未使用索引时的查询情况，EXPLAIN 语句执行如下：

```
mysql> EXPLAIN SELECT * FROM fruits WHERE f_name='apple';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
rows | Extra      |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | fruits | ALL | NULL | NULL | NULL | NULL | 15 | Using
where |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

可以看到，rows 列的值是 15，说明 “SELECT * FROM fruits WHERE f_name='apple';” 这个查询语句扫描了表中的 15 条记录。

然后，在 fruits 表的 f_name 字段上加上索引。执行添加索引的语句及结果如下：

```
mysql> CREATE INDEX index_name ON fruits(f_name);
```

```
Query OK, 0 rows affected (0.04 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

现在，再分析上面的查询语句。执行的 EXPLAIN 语句及结果如下：

```
mysql> EXPLAIN SELECT * FROM fruits WHERE f_name='apple';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | fruits | ref | index_name | index_name | 255 | const | 1 |
+----+-----+-----+-----+-----+-----+-----+-----+
Using where |
+----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

结果显示，rows 列的值为 1。这表示这个查询语句只扫描了表中的一条记录，其查询速度自然比扫描 15 条记录快。而且 possible_keys 和 key 的值都是 index_name，这说明查询时使用了 index_name 索引。

7.6.2 如何使用索引查询

索引可以提高查询的速度。但并不是使用带有索引的字段查询时，索引都会起作用。本小节将向读者介绍索引的使用。

使用索引有几种特殊情况，在这些情况下，有可能使用带有索引的字段查询时，索引并没有起作用，下面重点介绍这几种特殊情况。

1. 使用 LIKE 关键字的查询语句

在使用 LIKE 关键字进行查询的查询语句中，如果匹配字符串的第一个字符为“%”，索引不会起作用。只有“%”不在第一个位置，索引才会起作用。下面将举例说明。

查询语句中使用 LIKE 关键字，并且匹配的字符串中含有“%”字符，EXPLAIN 语句执行如下：

```
mysql> EXPLAIN SELECT * FROM fruits WHERE f_name like '%x';
+----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | fruits | ALL | NULL | NULL | NULL | 16 | Using where |
+----+-----+-----+-----+-----+-----+-----+-----+
```

```

+---+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM fruits WHERE f_name like 'x%';
+---+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+---+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | fruits | range | index_name | index_name | 150 | NULL | 4 |
Using where |
+---+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

已知 `f_name` 字段上有索引 `index_name`。第 1 个查询语句执行后，`rows` 列的值为 16，表示这次查询过程中扫描了表中所有的 16 条记录；第 2 个查询语句执行后，`rows` 列的值为 4，表示这次查询过程扫描了 4 条记录。第 1 个查询语句索引没有起作用，因为第 1 个查询语句的 `LIKE` 关键字后的字符串以 “%” 开头，而第 2 个查询语句使用了索引 `index_name`。

2. 使用多列索引的查询语句

MySQL 可以为多个字段创建索引。一个索引可以包括 16 个字段。对于多列索引，只有查询条件中使用了这些字段中第 1 个字段时，索引才会被使用。

本例在表 `fruits` 中 `f_id`、`f_price` 字段创建多列索引，验证多列索引的使用情况。

```

mysql> CREATE INDEX index_id_price ON fruits(f_id, f_price);
Query OK, 0 rows affected (0.39 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> EXPLAIN SELECT * FROM fruits WHERE f_id='12';
+---+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref |
+---+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | fruits | const | PRIMARY,index_id_price | PRIMARY | 20 | const | 1 |
+---+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> EXPLAIN SELECT * FROM fruits WHERE f_price=5.2;

```



```

+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | type | possible_keys | key | key_len | ref | rows |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | fruits | ALL | NULL | NULL | NULL | NULL | 16 | Using where |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

```

从第1条语句查询结果可以看出,“f_id='12'”的记录有1条。第1条语句共扫描了1条记录,并且使用了索引 index_id_price。从第2条语句查询结果可以看出,rows 列的值是16,说明查询语句共扫描了16条记录,并且key 列值为NULL,说明“SELECT * FROM fruits WHERE f_price=5.2;”语句并没有使用索引。因为f_price 字段是多列索引的第2个字段,只有查询条件中使用了f_id 字段才会使index_id_price 索引起作用。

3. 使用 OR 关键字的查询语句

查询语句的查询条件中只有 OR 关键字,且 OR 前后的两个条件中的列都是索引时,查询中才使用索引。否则,查询将不使用索引。

查询语句使用 OR 关键字的情况:

```

mysql> EXPLAIN SELECT * FROM fruits WHERE f_name='apple' or s_id=101 \G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: fruits
        type: ALL
possible_keys: index_name
         key: NULL
      key_len: NULL
         ref: NULL
        rows: 16
      Extra: Using where
1 row in set (0.00 sec)

```

```

mysql> EXPLAIN SELECT * FROM fruits WHERE f_name='apple' or f_id='12' \G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: fruits
        type: index_merge
possible_keys: PRIMARY,index_name,index_id_price
         key: index_name,PRIMARY

```

```

key_len: 510,20
  ref: NULL
  rows: 2
  Extra: Using union(index_name,PRIMARY); Using where
1 row in set (0.00 sec)

```

因为 `s_id` 字段上没有索引，第 1 条查询语句没有使用索引，总共查询了 16 条记录；第 2 条查询语句使用了 `f_name` 和 `f_id` 这两个索引，因为 `title` 字段和 `name` 字段上都有索引，查询的记录数为 2 条。

7.7 不同类型 SQL 语句优化方法

本节将针对不同类型的 SQL 语句进行一些优化的操作方法。

7.7.1 优化 INSERT 语句

当进行数据 INSERT 的时候，MySQL 客户端大致要经过的步骤如下：

- 客户端连接 MySQL 服务器。
- 客户端发送 INSERT 语句到服务器。
- 服务器解析 INSERT 语句。
- 服务器增加数据。
- 服务器给增加的记录添加索引。
- 服务器关闭连接。

优化 INSERT 语句的常见方法有以下几种。

(1) 根据以上步骤可以看出，如果从同一客户端插入很多行数据到 MySQL 服务器，如果一次性插入多个值，将大大缩短客户端与数据库服务器之间的连接和关闭等操作，例如：

```

mysql> insert into books values (1,'book1'), (2,'book2'), (3,'book3');
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

```

当从一个文本文件装载一个表时，使用 `LOAD DATA INFILE` 加载数据往往比使用很多 INSERT 语句效率至少提高 20 倍。

(2) 对于 MyISAM 类型的表，如果从不同客户插入很多行，可以通过使用 `INSERT DELAYED` 语句提升执行速度。`INSERT DELAYED INTO` 是客户端提交数据给 MySQL 服务器，MySQL 服务器返回 OK 状态给客户端，而这并不是将数据立即执行插入到表，而是存储在内存里面等待排队，直到当 MySQL 服务器有空闲时再插入。此时数据并没有真正写入磁盘，这

样的好处是提高插入数据的速度，不好的地方在于，如果系统崩溃，MySQL 还没有写入磁盘的数据将会丢失。

(3) 通常可以锁定表以加速插入数据，命令如下：

```
mysql> lock tables test write;
Query OK, 0 rows affected (0.03 sec)
mysql> insert into test values(3,'t3'),(4,'t4');
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

如果不加锁定表，每一次执行 INSERT 语句完成后，索引缓冲区都会被写到磁盘上，而加入锁定后索引缓冲区仅被写到磁盘上 1 次。

7.7.2 优化 ORDER BY 语句

在某些情况下，MySQL 可以使用一个索引来满足 ORDER BY 子句，而不需要额外的排序。

通常可以采用索引来对 ORDER BY 语句进行优化。下面通过一个案例来理解。

首先，先看一个简单的语句，命令如下：

```
mysql> select id,ename,depno,sal from emp where depno = 10;
```

在该语句中，如果对 id、ename、sal 字段添加索引，意义并不大，对这个语句没有任何优化作用，但是如果在外键 depno 上添加一个索引，优化的效果就很明显了。

下面介绍几个常用优化 ORDER BY 语句的例子：

(1) 对 ORDER BY + LIMIT 组合的索引优化，SQL 形式如下：

```
SELECT [column1]...FROM [TABLE] ORDER BY [sort] LIMIT[offset],[LIMIT];
```

该 SQL 语句优化只需要在[sort]上建立索引即可。

(2) 对 WHERE+ORDER BY + LIMIT 组合的索引优化，SQL 形式如下：

```
SELECT [column1]...FROM [TABLE] WHERE [columnX]=[value] ORDER BY [sort]
LIMIT[offset],[LIMIT];
```

此时如果只对[sort]添加索引，效率不是很高，还可以采用更加高效的方法建立一个联合索引(columnX,sort)。

(3) 不要对 WHERE 和 ORDER BY 的选项使用表达式或者函数，SQL 形式如下：

```
SELECT * FROM [TABLE] ORDER BY YEAR(date) LIMIT 0,30;
```


下面几种情况下不应该使用索引：

(1) order by 的字段混合使用 asc 和 desc。

```
mysql> select *from ordertable order by col1 desc,col2 asc;
```

(2) where 子句使用的字段和 order by 的字段不一致。

```
mysql> select *from ordertable where col1 = 1 order by col2;
```

(3) 对不同的关键字使用 order by 排序。

```
mysql> select *from ordertable order by col1,col2;
```

7.7.3 优化 GROUP BY 语句

使用 GROUP BY 语句时，MySQL 会对符合的结果自动排序。通过扫描整个表并创建一个新的临时表，表中每个组的所有行应为连续的，然后使用该临时表来找到组并应用累计行数。在某些情况下，MySQL 可以通过索引访问而不用临时表。

通过指定 ORDER BY NULL 可以禁止排序，从而可以节省耗损。下面通过案例来理解。首先分析没有优化的 GROUP BY 语句，命令如下：

```
mysql> explain select id,count(data) from test group by id \G;
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: test
         type: ALL
possible_keys: NULL
          key: NULL
       key_len: NULL
         ref: NULL
        rows: 4
   Extra: Using temporary; Using filesort
1 row in set (0.00 sec)
```

然后通过 ORDER BY NULL 优化 GROUP BY 语句，分析命令如下：

```
mysql> explain select id,count(data) from test group by id order by null\G;
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: test
         type: ALL
possible_keys: NULL
          key: NULL
       key_len: NULL
         ref: NULL
```

```

      rows: 4
      Extra: Using temporary
1 row in set (0.00 sec)

```

从执行计划可以看到，第一个 SQL 语句使用了“Using filesort”，而使用了 ORDER BY NULL 的 GROUP BY 语句减少了文件排序的步骤，当返回结果集很大时，对于 GROUP BY 的性能是有很大改善。

7.7.4 优化嵌套查询

MySQL 从 4.1 版本开始支持子查询，使用子查询可以进行 SELECT 语句的嵌套查询，即一个 SELECT 查询的结果作为另一个 SELECT 语句的条件。子查询可以一次性完成很多逻辑上需要多个步骤才能完成的 SQL 操作。子查询虽然可以使查询语句很灵活，但执行效率不高。执行子查询时，MySQL 需要为内层查询语句的查询结果建立一个临时表。然后外层查询语句从临时表中查询记录。查询完毕后，再撤销这些临时表。因此，子查询的速度会受到一定的影响。如果查询的数据量比较大，这种影响就会随之增大。

在 MySQL 中，可以使用连接（JOIN）查询来替代子查询。连接查询不需要建立临时表，其速度比子查询要快，如果查询中使用索引的话，性能会更好。连接之所以更有效率，是因为 MySQL 不需要在内存中创建临时表来完成查询工作。

先分析下面的子查询语句：

```

mysql> explain select * from emp where dept_id not in(select id from dept) \G;
***** 1. row *****
      id: 1
  select_type: PRIMARY
      table: emp
      type: ALL
possible_keys: NULL
      key: NULL
     key_len: NULL
      ref: NULL
      rows: 32768
     Extra: Using where
***** 2. row *****
      id: 2
  select_type: DEPENDENT SUBQUERY
      table: dept
      type: index_subquery
possible_keys: index_dept_id
      key: index_dept_id
     key_len: 5
      ref: func
      rows: 4096

```

```
Extra: Using index; Full scan on NULL key
2 rows in set (0.00 sec)
```

上面的子查询可以被 JOIN 代替，分析命令如下。

```
mysql> explain select * from emp left join dept on emp.dept_id = dept.id where
emp.dept_id is null \G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
       type: ALL
possible_keys: NULL
      key: NULL
     key_len: NULL
       ref: NULL
      rows: 32768
    Extra: Using where
***** 2. row *****
      id: 1
    select_type: SIMPLE
      table: dept
       type: ref
possible_keys: index_dept_id
      key: index_dept_id
     key_len: 5
       ref: test.emp.dept_id
      rows: 2048
    Extra: Using index
2 rows in set (0.00 sec)
```

连接 JOIN 之所以更有效率一些，主要是因为 MySQL 不需要在内存中创建临时表来完成查询过程中所需要的两个步骤，从执行计划可以看出采用 JOIN 联接查询的扫描记录的范围有了很大的改善，性能有一定的提高。

7.7.5 优化 OR 条件

对于使用 OR 条件语句的子查询，如果要使用索引，则 OR 之间的每个条件列都必须使用到索引。如果没有索引，可以考虑添加索引。

下面我们看一个例子，首先创建一个表 `ortest`，添加若干条记录，命令如下。

```
CREATE TABLE `ortest` (
  `id` int(11) default NULL,
  `data1` int(11) default NULL,
  `data2` int(11) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```


下面使用 OR 条件查询，分析命令如下。

```
mysql> explain select * from ortest where data1=2 or data2 = 3 \G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: ortest
         type: ALL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
         rows: 33141
      Extra: Using where
1 row in set (0.00 sec)
```

接着，在 data1 列和 data2 列中添加条件索引，命令如下。

```
mysql> create index index_data1 on ortest(data1);
Query OK, 32768 rows affected (0.48 sec)
Records: 32768 Duplicates: 0 Warnings: 0

mysql> create index index_data2 on ortest(data2);
Query OK, 32768 rows affected (0.77 sec)
Records: 32768 Duplicates: 0 Warnings: 0
```

创建两个索引后，发现 MySQL 在处理包含 OR 子句查询中，是对 OR 的各个字段在查询结果之后再进行 UNION 操作，分析结果如下：

```
mysql> explain select * from ortest where data1=2 or data2 = 3 \G;
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: ortest
         type: index_merge
possible_keys: index_data1,index_data2
         key: index_data1,index_data2
        key_len: 5,5
         ref: NULL
         rows: 2
      Extra: Using union(index_data1,index_data2); Using where
1 row in set (0.00 sec)
```

7.7.6 优化插入记录的速度

插入记录时,影响插入速度的主要是索引、唯一性校验、一次插入记录条数等。根据这些情况,可以分别进行优化。本小节将为读者介绍优化插入记录速度的几种方法。

对于 MyISAM 引擎的表,常见的优化方法如下:

1. 禁用索引

对于非空表,插入记录时,MySQL 会根据表的索引对插入的记录建立索引。如果插入大量数据,建立索引会降低插入记录的速度。为了解决这种情况,可以在插入记录之前禁用索引,数据插入完毕后再开启索引。禁用索引的语句如下:

```
ALTER TABLE table_name DISABLE KEYS;
```

其中 `table_name` 是禁用索引表的表名。

重新开启索引的语句如下:

```
ALTER TABLE table_name ENABLE KEYS;
```

对于空表批量导入数据,则不需要进行此操作,因为 MyISAM 引擎的表是在导入数据之后才建立索引的。

2. 禁用唯一性检查

插入数据时,MySQL 会对插入的记录进行唯一性校验。这种唯一性校验也会降低插入记录的速度。为了降低这种情况对查询速度的影响,可以在插入记录之前禁用唯一性检查,等到记录插入完毕后再开启。禁用唯一性检查的语句如下:

```
SET UNIQUE_CHECKS=0;
```

开启唯一性检查的语句如下:

```
SET UNIQUE_CHECKS=1;
```

例如:当 `SET UNIQUE_CHECKS=1` 的时候:

```
mysql> load data infile '/home/mysql/data2.txt' into table temp_table2;  
Query OK, 1356345 rows affected (24.23 sec)  
Records: 1356345 Deleted: 0 Skipped: 0 Warnings: 0
```

当 `SET UNIQUE_CHECKS=0` 的时候:

```
mysql> load data infile '/home/mysql/data2.txt' into table temp_table2;  
Query OK, 1356345 rows affected (18.23 sec)  
Records: 1356345 Deleted: 0 Skipped: 0 Warnings: 0
```

从对比结果可以看出,禁用唯一性检查后插入数据耗费的时间比较少。

3. 使用批量插入

插入多条记录时,可以使用一条 INSERT 语句插入一条记录;也可以使用一条 INSERT 语句插入多条记录。插入一条记录的 INSERT 语句情形如下:

```
INSERT INTO fruits VALUES('x1', '101', 'mongo2', '5.6');
INSERT INTO fruits VALUES('x2', '101', 'mongo3', '5.6');
INSERT INTO fruits VALUES('x3', '101', 'mongo4', '5.6');
```

使用一条 INSERT 语句插入多条记录的情形如下:

```
INSERT INTO fruits VALUES
('x1', '101', 'mongo2', '5.6'),
('x2', '101', 'mongo3', '5.6'),
('x3', '101', 'mongo4', '5.6');
```

第2种情形的插入速度要比第1种情形快。

4. 使用 LOAD 命令批量导入

MySQL 批量导入数据的时候,可以采用 load 命令提高导入的速度,对于 MyISAM 存储引擎,可以通过以下方式快速地导入大量的数据。

```
ALTER TABLE tblname DISABLE KEYS;
loading the data
ALTER TABLE tblname ENABLE KEYS;
```

这两个命令用来打开或者关闭 MyISAM 表非唯一索引的更新。在导入大量的数据到一个空的 MyISAM 表时,默认就是先导入数据,然后才创建索引;而在导入大量数据到一个非空的 MyISAM 表时,通过以上命令的设置,可以提高导入数据的效率。

首先直接使用 load 命令将数据导入 MySQL, 命令如下:

```
mysql> load data infile '/home/mysql/data.txt' into table temp_table;
Query OK, 609307 rows affected (1 min 62.17 sec)
Records: 609307 Deleted: 0 Skipped: 0 Warnings: 0
```

而使用 ALTER TABLE tblname DISABLE KEYS 方式耗时 8.23 秒,效率提升比较明显,如下所示:

```
mysql> alter table temp_table disable keys;
Query OK, 0 rows affected (0.00 sec)

mysql> load data infile '/home/mysql/data.txt' into table temp_table;
Query OK, 609307 rows affected (8.23 sec)
Records: 609307 Deleted: 0 Skipped: 0 Warnings: 0

mysql> alter table temp_table enable keys;
Query OK, 0 rows affected (12.15 sec)
```

对于 InnoDB 存储引擎的表而言,可以通过如下几种方式提高 InnoDB 表的导入效率。

(1) 禁用唯一性检查

插入数据之前执行 `set unique_checks=0` 来禁止对唯一索引的检查, 数据导入完成之后再运行 `set unique_checks=1`。这个和 MyISAM 引擎的使用方法一样。

(2) 禁用外键检查

插入数据之前执行禁止对外键的检查, 数据插入完成之后再恢复对外键的检查。禁用外键检查的语句如下:

```
SET foreign_key_checks=0;
```

恢复对外键的检查语句如下:

```
SET foreign_key_checks=1;
```

(3) 禁止自动提交

在导入数据前执行 `SET AUTOCOMMIT=0`, 关闭自动提交功能, 在成功导入数据之后, 执行 `SET AUTOCOMMIT=1`, 恢复自动提交功能, 可以提高导入的效率。

例如: 当 `SET AUTOCOMMIT=1` 的时候:

```
mysql> load data infile '/home/mysql/data3.txt' into table temp_table3;
Query OK, 1356345 rows affected (24.23 sec)
Records: 1356345 Deleted: 0 Skipped: 0 Warnings: 0
```

当 `SET AUTOCOMMIT=0` 的时候:

```
mysql> load data infile '/home/mysql/data3.txt' into table temp_table3;
Query OK, 1356345 rows affected (20.23 sec)
Records: 1356345 Deleted: 0 Skipped: 0 Warnings: 0
```

7.8 优化数据库结构

一个好的数据库设计方案对于数据库的性能常常会起到事半功倍的效果。合理的数据库结构不仅可以使数据库占用更小的磁盘空间, 而且能够使查询速度更快。数据库结构的设计, 需要考虑数据冗余、查询和更新的速度、字段的数据类型是否合理等多方面的内容。本节将为读者介绍优化数据库结构的方法。

7.8.1 将字段很多的表分解成多个表

对于字段较多的表, 如果有些字段的使用频率很低, 可以将这些字段分离出来形成新表。因为当一个表的数据量很大时, 会由于使用频率低的字段的存在而变慢。本小节将为读者介绍这种优化表的方法。

假设会员表存储会员登录认证信息, 该表中有很多字段, 如 `id`、姓名、密码、地址、电

话、个人描述字段。其中地址、电话、个人描述等字段并不常用。可以将这些不常用字段分解出另外一个表。将这个表取名叫 `members_detail`。表中有 `member_id`、`address`、`telephone`、`description` 等字段。其中，`member_id` 是会员编号，`address` 字段存储地址信息，`telephone` 字段存储电话信息，`description` 字段存储会员个人描述信息。这样就把会员表分成两个表，分别为 `members` 表和 `members_detail` 表。

创建这两个表的 SQL 语句如下：

```
CREATE TABLE members (
  Id int(11) NOT NULL AUTO_INCREMENT,
  username varchar(255) DEFAULT NULL ,
  password varchar(255) DEFAULT NULL ,
  last_login_time datetime DEFAULT NULL ,
  last_login_ip varchar(255) DEFAULT NULL ,
  PRIMARY KEY (Id)
) ;
CREATE TABLE members_detail (
  member_id int(11) NOT NULL DEFAULT 0,
  address varchar(255) DEFAULT NULL ,
  telephone varchar(16) DEFAULT NULL ,
  description text
) ;
```

这两个表的结构如下：

```
mysql> desc members;
```

| Field | Type | Null | Key | Default | Extra |
|-----------------|--------------|------|-----|---------|----------------|
| Id | int(11) | NO | PRI | NULL | auto_increment |
| username | varchar(255) | YES | | NULL | |
| password | varchar(255) | YES | | NULL | |
| last_login_time | datetime | YES | | NULL | |
| last_login_ip | varchar(255) | YES | | NULL | |

```
5 rows in set (0.00 sec)
```

```
mysql> DESC members_detail;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|--------------|------|-----|---------|-------|
| member_id | int(11) | NO | PRI | 0 | |
| address | varchar(255) | YES | | NULL | |

```
| telephone | varchar(16) | YES | | NULL | |
| description | text | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

如果需要查询会员的详细信息，可以用会员的 id 来查询。如果需要将会员的基本信息和详细信息同时显示，可以将 members 表和 members_detail 表进行联合查询，查询语句如下：

```
SELECT * FROM members LEFT JOIN members_detail ON
members.id=members_detail.member_id;
```

通过这种分解，可以提高表的查询效率。对于字段很多且有些字段使用不频繁的表，可以通过这种分解的方式来优化数据库的性能。

7.8.2 增加中间表

对于需要经常联合查询的表，可以建立中间表以提高查询效率。通过建立中间表，把需要经常联合查询的数据插入到中间表中，然后将原来的联合查询改为对中间表的查询，以此来提高查询效率。本小节将为读者介绍增加中间表优化查询的方法。

首先，分析经常联合查询表中的字段。然后，使用这些字段建立一个中间表，并将原来联合查询的表的数据插入到中间表中。最后，可以使用中间表来进行查询了。

会员信息表和会员组信息表的 SQL 语句如下：

```
CREATE TABLE vip(
  Id int(11) NOT NULL AUTO_INCREMENT,
  username varchar(255) DEFAULT NULL,
  password varchar(255) DEFAULT NULL,
  groupId INT(11) DEFAULT 0,
  PRIMARY KEY (Id)
);
CREATE TABLE vip_group (
  Id int(11) NOT NULL AUTO_INCREMENT,
  name varchar(255) DEFAULT NULL,
  remark varchar(255) DEFAULT NULL,
  PRIMARY KEY (Id)
);
```

查询会员信息表和会员组信息表。

```
mysql> DESC vip;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Id    | int(11) | NO | PRI | NULL | auto_increment |
```



```
| username | varchar(255) | YES | | NULL |
| password | varchar(255) | YES | | NULL |
| groupId | int(11) | YES | | NULL |
```

```
4 rows in set (0.01 sec)
```

```
mysql> DESC vip_group;
```

```
| Field | Type          | Null | Key | Default | Extra          |
| Id    | int(11)       | NO   | PRI | NULL    | auto_increment |
| name  | varchar(255)  | YES  |     | NULL    |                |
| remark | varchar(255)  | YES  |     | NULL    |                |
```

```
3 rows in set (0.01 sec)
```

已知现在有一个模块需要经常查询带有会员组名称、会员组备注 (remark)、会员用户名信息的会员信息。根据这种情况可以创建一个 temp_vip 表。temp_vip 表中存储用户名 (user_name)，会员组名称 (group_name) 和会员组备注 (group_remark) 信息。创建表的语句如下：

```
CREATE TABLE temp_vip (
  Id int(11) NOT NULL AUTO_INCREMENT,
  user_name varchar(255) DEFAULT NULL,
  group_name varchar(255) DEFAULT NULL,
  group_remark varchar(255) DEFAULT NULL,
  PRIMARY KEY (Id)
);
```

接下来，从会员信息表和会员组表中查询相关信息存储到临时表中：

```
mysql> INSERT INTO temp_vip(user_name, group_name, group_remark)
-> SELECT v.username,g.name,g.remark
-> FROM vip as v ,vip_group as g
-> WHERE v.groupId =g.Id;
Query OK, 0 rows affected (0.95 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

以后，可以直接从 temp_vip 表中查询会员名、会员组名称和会员组备注，而不用每次都进行联合查询。这样可以提高数据库的查询速度。

7.8.3 增加冗余字段

设计数据库表时应尽量遵循范式理论的规约，尽可能减少冗余字段，让数据库设计看起来精致、优雅。但是，合理地加入冗余字段也可以提高查询速度。本小节将为读者介绍通过增加

冗余字段来优化查询速度的方法。

表的规范化程度越高，表与表之间的关系就越多，需要连接查询的情况也就越多。例如，员工的信息存储在 `staff` 表中，部门信息存储在 `department` 表中。通过 `staff` 表中的 `department_id` 字段与 `department` 表建立关联关系。如果要查询一个员工所在部门的名称，必须从 `staff` 表中查找员工所在部门的编号（`department_id`），然后根据这个编号去 `department` 表查找部门的名称。如果经常需要进行这个操作，连接查询会浪费很多时间。可以在 `staff` 表中增加一个冗余字段 `department_name`，该字段用来存储员工所在部门的名称，这样就不用每次都进行连接操作了。

提示

冗余字段会导致一些问题。比如，冗余字段的值在一个表中被修改了，就要想办法在其他表中更新该字段。否则就会使原本一致的数据变得不一致。分解表、增加中间表和增加冗余字段都浪费了一定的磁盘空间。从数据库性能来看，为了提高查询速度而增加少量的冗余大部分时候是可以接受的；是否通过增加冗余来提高数据库性能，这要根据实际需求综合分析。

7.9 分析表、检查表和优化表

MySQL 提供了分析表、检查表和优化表的语句。分析表主要是分析关键字的分布；检查表主要是检查表是否存在错误；优化表主要是消除删除或者更新造成的空间浪费。本小节将为读者介绍分析表、检查表和优化表的方法。

7.9.1 分析表

MySQL 中提供了 `ANALYZE TABLE` 语句分析表，`ANALYZE TABLE` 语句的基本语法如下：

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name[,tbl_name]...
```

`LOCAL` 关键字是 `NO_WRITE_TO_BINLOG` 关键字的别名，二者都是执行过程不写入二进制日志，`tbl_name` 为分析的表的表名，可以有一个或多个。

使用 `ANALYZE TABLE` 分析表的过程中，数据库系统会自动对表加一个只读锁。在分析期间，只能读取表中的记录，不能更新和插入记录。`ANALYZE TABLE` 语句能够分析 `InnoDB`、`BDB` 和 `MyISAM` 类型的表。

使用 `ANALYZE TABLE` 来分析 `message` 表，执行的语句及结果如下：

```
mysql> ANALYZE TABLE message;
+-----+-----+-----+-----+
| Table      | Op      | Msg_type | Msg_text |
+-----+-----+-----+-----+
```

```

+-----+-----+-----+-----+
| test.fruits | analyze | status | OK      |
+-----+-----+-----+-----+
1 row in set (0.18 sec)

```

上面结果显示的信息说明如下：

- Table: 表示分析的表的名称。
- Op: 表示执行的操作。analyze 表示进行分析操作。
- Msg_type: 表示信息类型，其值通常是状态（status）、信息（info）、注意（note）、警告（warning）和错误（error）之一。
- Msg_text: 显示信息。

7.9.2 检查表

MySQL 中可以使用 CHECK TABLE 语句来检查表。CHECK TABLE 语句能够检查 InnoDB 和 MyISAM 类型的表是否存在错误。对于 MyISAM 类型的表，CHECK TABLE 语句还会更新关键字统计数据。而且，CHECK TABLE 也可以检查视图是否有错误，比如在视图定义中被引用的表已不存在。该语句的基本语法如下：

```

CHECK TABLE tbl_name [, tbl_name] ... [option] ...
option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}

```

其中，tbl_name 是表名；option 参数有 5 个取值，分别是 QUICK、FAST、MEDIUM、EXTENDED 和 CHANGED。各个选项的意义分别是：

- QUICK: 不扫描行，不检查错误的连接。
- FAST: 只检查没有被正确关闭的表。
- CHANGED: 只检查上次检查后被更改的表和没有被正确关闭的表。
- MEDIUM: 扫描行，以验证被删除的连接是有效的。也可以计算各行的关键字校验和，并使用计算出的校验和验证这一点。
- EXTENDED: 对每行的所有关键字进行一个全面的关键字查找。这可以确保表是 100% 一致的，但是花的时间较长。

option 只对 MyISAM 类型的表有效，对 InnoDB 类型的表无效。CHECK TABLE 语句在执行过程中也会给表加上只读锁。

7.9.3 优化表

MySQL 中使用 OPTIMIZE TABLE 语句来优化表。该语句对 InnoDB 和 MyISAM 类型的表都有效。但是，OPTIMIZE TABLE 语句只能优化表中的 VARCHAR、BLOB 或 TEXT 类型的字段。OPTIMIZE TABLE 语句的基本语法如下：

```

OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...

```


LOCAL | NO_WRITE_TO_BINLOG 关键字的意义和分析表相同，都是指定不写入二进制日志；tbl_name 是表名。

通过 OPTIMIZE TABLE 语句可以消除删除和更新造成的文件碎片。OPTIMIZE TABLE 语句在执行过程中也会给表加上只读锁。

提示

一个表使用了 TEXT 或者 BLOB 这样的数据类型，如果已经删除了表的一大部分，或者已经对含有可变长度行的表（含有 VARCHAR，BLOB 或 TEXT 列的表）进行了很多更新，则应使用 OPTIMIZE TABLE 来重新利用未使用的空间，并整理数据文件的碎片。在多数的设置中，根本不需要运行 OPTIMIZE TABLE。即使对可变长度的行进行了大量的更新，也不需要经常运行，每周一次或每月一次即可，并且只需要对特定的表运行。

7.10 小结

本章主要讲解了 SQL 优化的基本思路，在对 SQL 性能分析的时候要充分利用 EXPLAIN 和 Profiling 工具进行分析，然后在数据库表中合理地添加索引可以更好地提高 SQL 执行的效率。本章详细讲解了如何使用索引，另外对 SQL 通常执行的语句的优化方法进行了详细的讲解。最后讲解了数据库结构的优化方法。通过本章的学习可以更好地对 SQL 语句进行优化，从而提升 MySQL 数据库的整体性能。

第 8 章

◀ MySQL 服务器性能优化 ▶

和大多数数据库一样，MySQL 提供了很多的参数来进行服务器的优化设置，数据库服务器第一次启动的时候，很多参数都是默认设置的，这在实际生产环境中并不能完全满足需求，为此数据库管理员要进行必要的设置。本章主要讲解了 MySQL 服务器优化方面的一些知识和技巧，其中包括 MySQL 数据库源码安装优化、MySQL 服务器配置优化、I/O 调优，以及其他的一些 MySQL 优化技巧。

8.1 MySQL 源码安装的性能优化

很多熟悉 MySQL 的用户都喜欢使用源码包来进行安装，因为在安装源码的过程中可以非常方便地进行性能的优化，下面就源码安装过程中涉及的优化项进行简单地介绍，具体操作步骤如下：

步骤 01 首先下载 mysql-5.5.24.tar.gz 和 cmake-2.8.4.tar.gz 两个源文件。

步骤 02 安装 MySQL 源文件包是采用 cmake 安装的，首先安装 cmake。

```
[root@localhost Mysql5.5]# tar -zxv -f cmake-2.8.4.tar.gz
[root@localhost Mysql5.5]# cd cmake-2.8.4
[root@localhost cmake-2.8.4]# ./configure
-----
CMake 2.8.4, Copyright 2000-2009 Kitware, Inc.
Found GNU toolchain
...
[root@localhost Mysql5.5]# make
[root@localhost Mysql5.5]# make install
```

步骤 03 创建 MySQL 安装程序的目录和数据文件的目录。

```
[root@localhost ~]# mkdir -p /usr/local/mysql
[root@localhost ~]# mkdir -p /usr/local/mysql/data
```

步骤 04 创建 mysql 用户和 mysql 用户组。

```
[root@localhost ~]# groupadd mysql
```

```
[root@localhost ~]# useradd -r -g mysql mysql
```

步骤 05 解压缩 MySQL 源代码，并用 cmake 安装 MySQL 源代码。

```
[root@localhost Mysql5.5]# tar -zxv -f mysql-5.5.24.tar.gz
[root@localhost Mysql5.5]# cd mysql-5.5.24/
[root@localhost mysql-5.5.24]# cmake -DCMAKE_INSTALL_PREFIX=/usr/local/mysql
-DMYSQL_DATADIR=/usr/local/mysql/data -DDEFAULT_CHARSET=utf8
-DDEFAULT_COLLATION=utf8_general_ci -DEXTRA_CHARSETS=all
-DENABLED_LOCAL_INFILE=1 -DWITH_INNOBASE_STORAGE_ENGINE=1
-DWITH_ARCHIVE_STORAGE_ENGINE=1 -DWITH_BLACKHOLE_STORAGE_ENGINE=1
-DWITH_EXAMPLE_STORAGE_ENGINE=1 -DWITH_FEDERATED_STORAGE_ENGINE=1
-DWITH_PARTITION_STORAGE_ENGINE=1
[root@localhost mysql-5.5.24]# make
[root@localhost mysql-5.5.24]# make install
```

值得注意的是 cmake 后面紧跟着参数列表，而没有当前目录的点，参数和参数之间空格隔开，下面介绍主要参数的含义。

- DCMAKE_INSTALL_PREFIX=/usr/local/mysql: 表示 MySQL 源文件安装路径。
- DMYSQL_DATADIR=/usr/local/mysql/data: 表示 MySQL 数据文件的安装路径。
- DDEFAULT_CHARSET=utf8: 表示数据库默认的数据字符集。
- DDEFAULT_COLLATION=utf8_general_ci: 表示数据库默认的校验字符集。
- DEXTRA_CHARSETS=all: 表示数据库安装所有的扩展字符集。
- DENABLED_LOCAL_INFILE=1: 表示允许从本地导入数据。

步骤 06 设置目录权限。

```
[root@localhost mysql-5.5.24]# cd /usr/local/mysql/
[root@localhost mysql]# chown -R root:mysql .
[root@localhost mysql]# chown -R mysql:mysql data
```

步骤 07 创建 my.cnf 文件，并且开始初始化数据库。

```
[root@localhost mysql]# cp ./support-files/my-medium.cnf /etc/my.cnf
cp: overwrite '/etc/my.cnf'? y
[root@localhost mysql]# ./scripts/mysql_install_db --user=mysql
Installing MySQL system tables...
OK
```

步骤 08 设置环境变量。

```
[root@localhost mysql]# cd ~
[root@localhost ~]# vi .bash_profile
PATH=$PATH:$HOME/bin:/usr/local/mysql/bin:/usr/local/mysql/lib
```

步骤 09 启动 MySQL。


```
[root@localhost ~]# cd /usr/local/mysql/
[root@localhost mysql]# ./bin/mysqld_safe --user=mysql &
[1] 4833
[root@localhost mysql]# 120915 11:00:42 mysqld_safe Logging to
'/usr/local/mysql/data/localhost.localdomain.err'.
120915 11:00:42 mysqld_safe Starting mysqld daemon with databases from
/usr/local/mysql/data
```

至此，源码安装成功。通过源码安装的方法，可以灵活地进行数据库的定制编译，因此有更强的灵活性，而且某些编译选项还可以提高数据的性能。

执行以下命令可以看到所有编译的配置选项，结果如下：

```
[root@localhost cmake-2.8.4]# ./configure --help
Usage: /root/MySQL5.5/cmake-2.8.4/bootstrap [options]
Options: [defaults in brackets after descriptions]
Configuration:
  --help                print this message
  --version              only print version information
  --verbose              display more information
  --parallel=n           bootstrap cmake in parallel, where n is
                        number of nodes [1]
  --enable-ccache        Enable ccache when building cmake
  --init=FILE            load FILE as script to populate cache
  --system-libs          use all system-installed third-party libraries
                        (for use only by package maintainers)
  --no-system-libs       use all cmake-provided third-party libraries
                        (default)
  --system-curl          use system-installed curl library
  --no-system-curl       use cmake-provided curl library (default)
  --system-expat         use system-installed expat library
  --no-system-expat     use cmake-provided expat library (default)
  --system-zlib          use system-installed zlib library
  --no-system-zlib       use cmake-provided zlib library (default)
  --system-bzip2         use system-installed bzip2 library
  --no-system-bzip2     use cmake-provided bzip2 library (default)
  --system-libarchive    use system-installed libarchive library
  --no-system-libarchive use cmake-provided libarchive library (default)

  --qt-gui               build the Qt-based GUI (requires Qt >= 4.2)
  --no-qt-gui            do not build the Qt-based GUI (default)
  --qt-qmake=<qmake>     use <qmake> as the qmake executable to find Qt

Directory and file names:
  --prefix=PREFIX        install files in tree rooted at PREFIX
                        [{cmake_default_prefix}]
  --datadir=DIR           install data files in PREFIX/DIR
```

```

                                [/share/CMake]
--docdir=DIR                    install documentation files in PREFIX/DIR
                                [/doc/CMake]
--mandir=DIR                    install man pages files in PREFIX/DIR/manN
                                [/man]

```

下面通过改变编译选项，提高数据的性能。具体修改如下：

(1) 可以使用如下命令来指定 MySQL 数据的源文件和数据文件存放的位置，改变安装前缀，将所有安装的内容安装到/usr/local/mysql 5.5 目录下。

```
[root@localhost cmake-2.8.4]# ./configure --prefix=/usr/local/mysql5.5
```

(2) 可以通过修改 with-unix-socket-path 的参数选项来改变数据库 sock 文件的位置。

```
[root@localhost cmake-2.8.4]#
./configure --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

(3) MySQL 数据库使用 LATIN1 和 LANTIN1_SWEDISH_CI 作为默认的字符集和校对规则。如果想改变安装后的默认字符集和默认排序规则，可以使用如下编译选项，命令如下：

```
[root@localhost cmake-2.8.4]# ./configure --with-charset=CHARSET
[root@localhost cmake-2.8.4]# ./configure --with-collation=COLLATION
```

(4) MySQL 数据库源码安装可以采用静态编译以提高性能，命令如下：

```
[root@localhost cmake-2.8.4]# ./configure --with-client-ldflags=-all-static
--with-mysqld-ldflags=-all-static
```

上述参数的含义如下。

- --with-client-ldflags=-all-static: 表示以静态编译方式编译客户端。
- --with-mysqld-ldflags=-all-static: 表示以静态编译方式编译服务端。

8.2 MySQL 服务器配置优化

在 MySQL 数据库系统中，有些参数直接影响到系统的整体性能。MySQL 默认的参数设置往往不能达到实际工作的要求，此时需要通过调整服务器的一些参数来满足实际应用的需求。本小节主要介绍一些 MySQL 服务器的参数调整方法来优化 MySQL 服务器的性能，这里是一些通用的方法，不涉及具体的存储引擎。

8.2.1 查看性能参数的方法

MySQL 服务启动后，可以使用 SHOW VARIABLES 语句来查询服务器一些静态的参数，比如，缓冲区大小、字符集、数据文件名称等信息，命令如下：

```
mysql> show variables;
```

| Variable_name | Value |
|--------------------------|----------------------------------------------|
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| autocommit | ON |
| automatic_sp_privileges | ON |
| back_log | 50 |
| basedir | /root/tools/mysql-5.1.30-linux-i686-glibc23/ |
| big_tables | OFF |
| binlog_cache_size | 32768 |
| binlog_format | STATEMENT |
| bulk_insert_buffer_size | 8388608 |
| character_set_client | utf8 |
| ... | ... |

267 rows in set (0.08 sec)

SHOW VARIABLES 查看的是 MySQL 启动之前已经配置好的一些系统静态参数。使用 SHOW STATUS 命令查询服务器运行中的状态信息，比如，当前连接数、锁等待状态信息，如下所示。

```
mysql> show status;
```

| Variable_name | Value |
|------------------------|-------|
| Aborted_clients | 0 |
| Aborted_connects | 0 |
| Binlog_cache_disk_use | 0 |
| Binlog_cache_use | 0 |
| Bytes_received | 1567 |
| Bytes_sent | 24957 |
| Com_admin_commands | 0 |
| Com_assign_to_keycache | 0 |
| Com_alter_db | 0 |
| Com_alter_db_upgrade | 0 |
| Com_alter_event | 0 |
| Com_alter_function | 0 |
| Com_alter_procedure | 0 |
| Com_alter_server | 0 |
| Com_alter_table | 0 |
| Com_alter_tablespace | 0 |


```

| Com_analyze           | 0      |
| Com_backup_table      | 0      |
| Com_begin             | 0      |
| Com_binlog            | 0      |
| Com_call_procedure    | 0      |
| Com_change_db         | 0      |
| Com_change_master     | 0      |
| ...                   | ...    |
+-----+-----+
290 rows in set (0.80 sec)

```

同时，可以在操作系统下直接查询数据库的状态，命令如下。

```

[root@localhost ~]#mysqladmin -uroot variables
+-----+-----+
--+
| Variable_name          | Value                                     |
+-----+-----+
--+
| auto_increment_increment | 1                                     |
| auto_increment_offset   | 1                                     |
| autocommit              | ON                                    |
| automatic_sp_privileges | ON                                    |
| back_log                | 50                                    |
| basedir                 | /root/tools/mysql-5.1.30-linux-i686-glibc23/ |
| big_tables              | OFF                                   |
| binlog_cache_size       | 32768                                |
| binlog_format           | STATEMENT                             |
| bulk_insert_buffer_size | 8388608                               |
| character_set_client    | utf8                                  |
| ...                     | ...                                   |
+-----+-----+
--+
267 rows in set (0.08 sec)

```

MySQL 服务器参数比较多，如果需要了解某个参数的含义，可以通过如下命令查询，方法如下。

```

[root@localhost ~]#mysqld -verbose -help|more
mysqld Ver 5.0.89-log for pc-linux-gnu on i686 (MySQL Community Server (GPL))
Copyright (C) 2000 MySQL AB, by Monty and others
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Starts the MySQL database server

Usage: mysqld [OPTIONS]

```

Default options are read from the following files in the given order:
 /etc/my.cnf /usr/local/mysql/etc/my.cnf ~/.my.cnf

The following groups are read: mysql_cluster mysqld server mysqld-5.0

The following options may be given as the first argument:

```
--print-defaults Print the program argument list and exit
--no-defaults      Don't read default options from any options file
--defaults-file=#  Only read default options from the given file #
--defaults-extra-file=# Read this file after the global files are read

-?, --help          Display this help and exit.
--abort-slave-event-count=#
                    Option used by mysql-test for debugging and testing of
                    replication.
--allow-suspicious-udfs
                    Allows use of UDFs consisting of only one symbol xxx()
                    without corresponding xxx_init() or xxx_deinit(). That
                    also means that one can load any function from any
                    library, for example exit() from libc.so
-a, --ansi          Use ANSI SQL syntax instead of MySQL syntax. This mode
                    will also set transaction isolation level 'serializable'.
--auto-increment-increment[=#]
                    Auto-increment columns are incremented by this
--auto-increment-offset[=#]
                    Offset added to Auto-increment columns. Used when
                    auto-increment-increment != 1
--automatic-sp-privileges
                    Creating and dropping stored procedures alters ACLs.
                    Disable with --skip-automatic-sp-privileges.
-b, --basedir=name  Path to installation directory. All paths are usually
                    resolved relative to this.
--bdb              Enable Berkeley DB (if this version of MySQL supports
                    it). Disable with --skip-bdb (will save memory).
```

```
...
sync_frm          TRUE
table_cache        64
table_lock_wait_timeout  50
thread_cache_size  0
thread_concurrency  10
thread_stack       196608
time_format        (No default value)
tmp_table_size     33554432
transaction_alloc_block_size  8192
transaction_prealloc_size  4096
updatable_views_with_limit  1
```



```
wait_timeout
```

```
28800
```

To see what values a running MySQL server is using, type 'mysqladmin variables' instead of 'mysql --verbose --help'.

从查询的结果可以看出，包含了当前服务器参数的介绍。如果需要查询其中的某些参数的含义，可以通过下面的命令过滤查询，查询结果如下。

```
[root@localhost ~]# mysql --verbose --help|grep thread
--log-slave-updates Tells the slave to log the updates from the slave thread
    The number of seconds the slave thread will sleep before
    set, the slave thread will not be started. Note that the
    master and where the I/O replication thread is in the
    The password the slave thread will authenticate with when
--master-user=name The username the slave thread will use for authentication
    for other threads.
    A dedicated thread is created to, at the given
    the SQL replication thread is in the relay logs.
    Tells the slave thread to restrict replication to the
    Tells the slave thread to restrict replication to the
    Tells the slave thread to not replicate to the specified
    Tells the slave thread to not replicate to the specified
    Tells the slave thread to restrict replication to the
    Tells the slave thread to not replicate to the tables
--skip-thread-priority
    Don't give threads different priorities.
    Tells the slave thread to continue replication when a
    have. This comes into play when the main MySQL thread
    that this is a limit per thread!
    How long a INSERT DELAYED thread should wait for INSERT
--flush_time=# A dedicated thread is created to flush all tables at the
    Number of times a thread is allowed to enter InnoDB
--innodb_file_io_threads=#
    Number of file I/O threads in InnoDB.
--innodb_thread_concurrency=#
    environments. Sets the maximum number of threads allowed
    inside InnoDB. Value 0 will disable the thread
--innodb_thread_sleep_delay=#
    Time of innodb thread sleeping before joining InnoDB
--max_delayed_threads=#
    Don't start more than this number of threads to handle
--myisam_repair_threads=#
    Number of threads to use when repairing MyISAM tables.
    Each thread that does a sequential scan allocates a
    exception for replication (slave) threads and users with
    Number of times the slave SQL thread will retry a
```



```

        If creating the thread takes longer than this value (in
        seconds), the Slow_launch_threads counter will be
        Each thread that needs to do a sort allocates a buffer of
--table_cache=#       The number of open tables for all threads.
--thread_cache_size=#
                        How many threads we should keep in a cache for reuse.
--thread_concurrency=#
                        Permits the application to give the threads system a hint
                        for the desired number of threads that should be run at
--thread_stack=#      The stack size for each thread.
innodb_file_io_threads      4
innodb_thread_concurrency   8
innodb_thread_sleep_delay   10000
max_delayed_threads         20
myisam_repair_threads       1
thread_cache_size           0
thread_concurrency          10
thread_stack                 196608

```

mysql --verbose --help|grep thread 命令会把包含 thread 的信息查询出来。了解查询 MySQL 服务器参数的方法之后，接下来介绍对 MySQL 数据库性能有重要影响的参数。在 MySQL 数据库中，key_buffer_size 和 table_cache 参数仅仅适用于 MyISAM 存储引擎。

8.2.2 key_buffer_size 的设置

在 MySQL 数据库中，key_buffer_size 参数是对 MyISAM 表性能影响最大的一个参数，先来看看 mysql 中是如何定义 key_buffer_size 参数的，如下所示：

```

[root@localhost ~]# mysql --verbose --help|grep "\-key_buffer_size" -A 5
--key_buffer_size=# The size of the buffer used for index blocks for MyISAM
                    tables. Increase this to get better index handling (for
                    all reads and multiple writes) to as much as you can
                    afford; 64M on a 256M machine that mainly runs MySQL is
                    quite common.

```

该参数用来设置索引块缓存的大小，只适用于 MyISAM 存储引擎，MySQL 5.1 以后提供了多个 key_buffer，可以将制定的表索引缓存到指定的 key_buffer，这样可以更好地降低线程之间的竞争。下面查询下 key_buffer_size 的值，如下所示。

```

mysql> show variables like 'key_buffer_size';
+-----+-----+
| Variable_name | Value      |
+-----+-----+
| key_buffer_size | 16777216 |
+-----+-----+
1 row in set (0.09 sec)

```

系统分配了索引缓冲区 16MB 的内存，下面修改该参数值到 200MB，如下所示。

```
mysql> set global key_buffer_size=204800;
Query OK, 0 rows affected (0.64 sec)

mysql> show variables like 'key_buffer_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_buffer_size | 204800 |
+-----+-----+
1 row in set (0.06 sec)
```

上面介绍的是默认的 `key_buffer`，下面介绍如何设置多个 `key_buffer`，先建立一个索引缓存，如下所示。

```
mysql> set global hot_cache2.key_buffer_size=128*1024;
Query OK, 0 rows affected (0.05 sec)
```

将相关表的索引放到指定的索引缓存中，下面将表 `t` 和 `t2` 的索引放到 `hot_cache2` 缓存中，如下所示。

```
mysql> cache index t,t2 in hot_cache2;
+-----+-----+-----+-----+
| Table | Op           | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

如果想要把表 `t` 的索引加载到默认的缓存(`key_buffer`)中，可以使用如下语句。

```
mysql> load index into cache t;
+-----+-----+-----+-----+
| Table | Op           | Msg_type | Msg_text |
+-----+-----+-----+-----+
| test.t | preload_keys | status   | OK       |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

如果要删除索引缓存，则只需要设置该缓存区大小为 0 即可，如下所示。

```
mysql> set global hot_cache2.key_buffer_size=0;
Query OK, 0 rows affected (0.00 sec)
```

值得注意的是，不能删除默认的索引缓存区，下面看下删除后的情况，如下所示。

```
mysql> show variables like 'key_buffer_size';
```

```

+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_buffer_size | 204800 |
+-----+-----+
1 row in set (0.02 sec)
mysql> set global key_buffer_size=0;
Query OK, 0 rows affected, 1 warning (0.05 sec)

```

接下来，查询 `key_buffer_size` 的大小，发现并没有删除成功，如下所示。

```

mysql> show variables like 'key_buffer_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_buffer_size | 204800 |
+-----+-----+
1 row in set (0.00 sec)

```

`Cache index` 命令可以将多个表的索引加载到指定的索引缓冲区中，但每次数据库重启后，索引缓冲区中的数据会清空，此时可以考虑在配置文件 `/etc/my.cnf` 中添加执行 `init-file` 选项，每次服务器启动的时候自动将指定表的索引加载到缓冲区中，如下所示。

```

[root@localhost ~]# more /etc/my.cnf
[mysqld]
port      = 3309
...
key_buffer_size = 1G
hot_cache.key_buffer_size = 512M
init_file = /root/initSQL/mysqld_init.sql
...

```

数据库每次启动的时候执行 `/root/initSQL/mysqld_init.sql` 脚本文件，该文件可以将多个表索引加载到缓存缓冲区中，下面的例子将表 `t` 和表 `t1` 的索引加载到 `hot_cache` 缓冲区中。

```
cache index t,t1 in hot_cache;
```

下面可以验证配置是否生效，如下所示。

```

mysql> show variables like 'key_buffer_size';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_buffer_size | 1073741824 |
+-----+-----+
1 row in set (0.01 sec)

mysql> select @@global.hot_cache.key_buffer_size;

```



```

+-----+
| @@global.hot_cache.key_buffer_size |
+-----+
|                               536870912 |
+-----+
1 row in set (0.00 sec)

```

8.2.3 table_cache 的设置

table_cache 参数与 key_buffer_size 参数一样只适用于 MyISAM 存储引擎，在 mysqld 中，对 table_cache 参数的定义如下：

```

[root@localhost ~]# mysqld --verbose --help|grep "\-table_cache"
--table_cache=#      The number of open tables for all threads.

```

该参数表示数据库用户打开表的缓存数量，每个连接进来，都会至少打开一个表缓存。因此该参数的值与 max_connections 有关。

当一个连接访问一个表时，如果该表已经在缓存中打开，则会直接访问缓存中的表信息，如果该表没有被缓存，则会将当前的表添加到缓存并进行查询。table_cache 用于限制缓存表的最大数目，如果当前缓存没有达到了 table_cache 缓存表数量的上限，则会将表缓存起来，如果达到了 table_cache 缓存表数量的上限，MySQL 将根据缓存表的最后查询时间、查询率等规则释放之前的缓存。

在设置该参数的时候，可以通过检查 open_tables 和 opened_tables 确定该参数的值，open_tables 参数表示当前打开的表缓存数，opened_tables 参数表示曾经打开的表缓存的数，如果执行 FLUSH TABLE 操作，则系统会关闭一些当前没有使用的表缓存，因此 FLUSH TABLE 操作后，open_tables 参数值会减少，opened_tables 参数值不会变。下面来验证这个过程。

步骤 01 清空缓存之前，记录 open_tables 和 opened_tables 的数量，如下所示。

```

mysql> show global status like 'open_tables';
+-----+
| Variable_name | Value |
+-----+
| Open_tables   | 9     |
+-----+
1 row in set (0.15 sec)

mysql> show global status like 'opened_tables';
+-----+
| Variable_name | Value |
+-----+
| Opened_tables | 16    |
+-----+
1 row in set (0.00 sec)

```

步骤 02 执行 FLUSH TABLES 操作，然后记录 open_tables 和 opened_tables 的数量，如下所示。

```
mysql> flush tables;
Query OK, 0 rows affected (0.00 sec)

mysql> show global status like 'open_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Open_tables   | 0     |
+-----+-----+
1 row in set (0.00 sec)

mysql> show global status like 'opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 16    |
+-----+-----+
1 row in set (0.00 sec)
```

此时 FLUSH TABLES 操作之后，open_tables 被清空了，而 opened_tables 没有发生改变。

步骤 03 执行一个 SQL 查询，如下所示。

```
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
|        10 |
+-----+
1 row in set (0.00 sec)
```

步骤 04 查询 open_tables 和 opened_tables 的数量，如下所示。

```
mysql> show global status like 'open_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Open_tables   | 1     |
+-----+-----+
1 row in set (0.00 sec)

mysql> show global status like 'opened_tables';
+-----+-----+
| Variable_name | Value |
```

```
+-----+-----+
| Opened_tables | 17 |
+-----+-----+
1 row in set (0.00 sec)
```

步骤 05 此时发现 `open_tables` 和 `opened_tables` 的数量都增加了 1, 此时继续对 `t1` 表进行查询, 如下所示。

```
mysql> select count(*) from t1;
+-----+
| count(*) |
+-----+
|      10 |
+-----+
1 row in set (0.00 sec)

mysql> show global status like 'open_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Open_tables   | 1     |
+-----+-----+
1 row in set (0.00 sec)

mysql> show global status like 'opened_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Opened_tables | 17    |
+-----+-----+
1 row in set (0.00 sec)
```

此时, 因为之前对表 `t` 查询时已经将表缓存到缓冲区了, 因此 `open_tables` 和 `opened_tables` 参数的值没有发生改变。因此, 在设置 `table_cache` 的时候, 可以参考 `open_tables` 的值。

8.2.4 内存参数的设置

对于数据库系统, 内存设置都会直接影响到系统的性能, 因为内存的读写速度要远远高于磁盘的速度, 所以应尽量让数据的读写直接在内存中进行。然而由于系统内存资源的限制, 过多地使用内存空间必将降低系统的整体性能, 所以内存的设置需要兼顾服务器的配置。

1. `innodb_buffer_pool_size` 的设置

在 `mysqld` 中, 对 `innodb_buffer_pool_size` 参数的定义如下:

```
[root@localhost ~]# mysql --verbose --help | grep "\-innodb_buffer_pool_size"
```



```
-A 2
```

```
--innodb_buffer_pool_size=#
```

The size of the memory buffer InnoDB uses to cache data and indexes of its tables.

该参数的作用是设置缓存 innodb 表和索引数据。这个值设置越高，访问表中数据需要的磁盘 I/O 越少。该参数的默认值是 128MB，建议不要将该参数设置过大，设置该参数应该遵循以下的分配原则。

- 如果数据库服务器是一个独立的服务器，可以考虑使用物理内存的 70%~80%。
- 由于该参数不能动态更改，要修改这个值，需要重启 mysqld 服务。
- 不要把该参数设置太大，如果分配过大，会导致操作系统的虚拟空间被占用，导致操作系统变慢，从而减低 SQL 查询的效率。

在设置该参数前，可以先查询实际内存参数的大小，如下所示。

```
mysql> show innodb status\G;
***** 1. row *****
Type: InnoDB
Name:
Status:
=====
120901 16:02:51 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 53 seconds
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 1, signal count 1
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 3, OS waits 1; RW-excl spins 0, OS waits 0
-----
TRANSACTIONS
-----
Trx id counter 0 2304
Purge done for trx's n:o < 0 1802 undo n:o < 0 0
History list length 6
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3271, OS thread id 3014364048
MySQL thread id 4, query id 19 192.168.0.119 user1
show innodb status
-----
FILE I/O
-----
I/O thread 0 state: waiting for i/o request (insert buffer thread)
```

```

I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
  ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
32 OS file reads, 3 OS file writes, 3 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s
-----

```

INSERT BUFFER AND ADAPTIVE HASH INDEX

```

-----
Ibuf: size 1, free list len 0, seg size 2,
0 inserts, 0 merged recs, 0 merges
Hash table size 34679, node heap has 0 buffer(s)
0.00 hash searches/s, 0.00 non-hash searches/s
---

```

LOG

```

---
Log sequence number 0 65767
Log flushed up to 0 65767
Last checkpoint at 0 65767
0 pending log writes, 0 pending chkp writes
8 log i/o's done, 0.00 log i/o's/second
-----

```

BUFFER POOL AND MEMORY

```

-----
Total memory allocated 17445316; in additional pool allocated 869760
Dictionary memory allocated 25752
Buffer pool size 512
Free buffers 488
Database pages 24
Modified db pages 0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 24, created 0, written 0
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
-----

```

ROW OPERATIONS

```

-----
0 queries inside InnoDB, 0 queries in queue
1 read views open inside InnoDB
Main thread process no. 3271, id 2978315152, state: waiting for server activity
Number of rows inserted 0, updated 0, deleted 0, read 0
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s

```

```
-----
END OF INNODB MONITOR OUTPUT
=====
```

```
1 row in set, 1 warning (0.41 sec)
```

从以上查询可以得出 Total memory allocated 17445316; in additional pool allocated 869760, 然后可以根据内存的大小来合理地设置该参数。

2. innodb_additional_mem_pool 的设置

在 mysqld 中, 对 innodb_additional_mem_pool 参数的定义如下:

```
[root@localhost~]# mysql --verbose --help|grep
"\-innodb_additional_mem_pool" -A 3
    --innodb_additional_mem_pool_size=#
                                Size of a memory pool InnoDB uses to store data
                                dictionary information and other internal data
                                structures.
```

这个参数用来设置 InnoDB 存储的数据目录信息和其他内部数据结构的内存池大小。应用程序里的表越多, 需要在这里分配越多的内存。对于一个相对稳定的应用, 这个参数的大小也是相对稳定的, 也没有必要预留非常大的值。如果 InnoDB 用光了这个池内的内存, InnoDB 开始从操作系统分配内存, 并且往 MySQL 错误日志写警告信息。默认值是 1MB, 当发现错误日志中已经有相关的警告信息时, 就应该适当地增加该参数的大小。

管理员也可以使用 show innodb status\G;命令查询运行中数据库的状态, 然后可以调整到合适的值, 如下所示。

```
mysql> show innodb status\G;
...
-----
BUFFER POOL AND MEMORY
-----
Total memory allocated 17445316; in additional pool allocated 869760
Dictionary memory allocated 25752
Buffer pool size      512
Free buffers          488
Database pages        24
Modified db pages     0
Pending reads         0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 24, created 0, written 0
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
...
```


8.2.5 日志和事务参数的设置

关系型数据库的日志与事务管理也是影响性能的重要因素之一，尤其是以事务为主的系统，日志读写比较频繁，进行合理的设置可以降低系统 I/O 和其他资源开销，提升系统性能。

1. innodb_log_file_size 的设置

在 mysql 中，对 innodb_log_file_size 参数的定义如下：

```
[root@localhost ~]# mysql --verbose --help|grep "\-innodb_log_file_size" -A
1
--innodb_log_file_size=#
    Size of each log file in a log group.
```

该参数的作用是设置日志组中每个日志文件的大小。该参数在高写入负载尤其是大数据集的情况下很重要，这个值越大则性能相对越高。该参数系统默认值是 5MB。该参数的设置应该遵循以下原则。

- 如果日志文件过小，日志切换比较频繁，会影响服务器性能，如果日志文件过大，当系统灾难时恢复时间会加大。
- 在 mysql 5.5 和 5.5 以前，innodb 的 logfile 最大设置为 4GB，在 5.6 以后的版本中，logfile 最大可以设为 512GB。一般大小控制在几个日志文件相加后在 2GB 以内最佳，具体情况还需要以数据大小为依据。

2. innodb_log_files_in_group 的设置

在 mysql 中，对 innodb_log_files_in_group 参数的定义如下：

```
[root@localhost ~]# mysql --verbose --help|grep
"\-innodb_log_files_in_group" -A 3
--innodb_log_files_in_group=#
    Number of log files in the log group. InnoDB writes to
    the files in a circular fashion. Value 3 is recommended
    here.
```

该参数用来指定数据库有几个日志组，默认为 2 个，因为有可能出现跨日志的大事务，所以一般来说，建议使用 3~4 个日志组。

3. innodb_log_buffer_size 的设置

在 mysql 中，对 innodb_log_buffer_size 参数的定义如下：

```
[root@localhost ~]# mysql --verbose --help|grep "\-innodb_log_buffer_size"
-A2
--innodb_log_buffer_size=#
    The size of the buffer which InnoDB uses to write log to
    the log files on disk.
```

该参数的作用是设置日志缓存的大小，一旦提交事务，则将该缓存池中的内容写到磁盘的日志文件上，该参数的设置在中等强度写入负载以及较短事务情况下，一般都可以满足服务器的性能要求。如果服务器负载较大，可以考虑加大该参数的值，一般缓存池中的内存每秒钟写到磁盘一次，所以设置该参数较大则会浪费内存空间，一般设置为 8MB~16MB 就足够了。

另外可以参考 `Innodb_os_log_written` 的值，如果这个值增加过快，可以适当地增加该参数的值。

4. `innodb_flush_log_at_trx_commit` 的设置

在 `mysqld` 中，对 `innodb_flush_log_at_trx_commit` 参数的定义如下：

```
[root@localhost~]#mysqld --verbose --help|grep
"\-\\-innodb_flush_log_at_trx_commit" -A 3
--innodb_flush_log_at_trx_commit[=#]
Set to 0 (write and flush once per second), 1 (write and
flush at each commit) or 2 (write at commit, flush once
per second).
```

该参数是用来控制缓冲区的数据写入到日志文件和将日志文件数据刷新到磁盘的操作时间。对该参数的设置可以在数据库性能与数据库安全之间进行折中。

- 当该参数设置为 0 时，日志缓存每秒一次地被写入到日志文件中，并且将日志文件数据刷新到磁盘。
- 当该参数设置为 1 时，InnoDB 的事务日志在每次提交后写入日志文件，并对日志做刷新到磁盘的操作。这个可以做到不丢失任何一个事务。
- 当该参数设置为 2 时，在每个事务提交时，日志缓冲被写到日志文件，但不对日志文件做到磁盘刷新的操作，对日志文件的刷新也是每秒发生一次。但需要注意的是，由于进程调度方面的问题，并不能保证日志文件的刷新操作每秒一定会发生。

`innodb_flush_log_at_trx_commit` 参数只有 3 个值，默认为 1，也是最安全的设置。处于性能考虑，可以设置为 0 或者 2，但会在数据库崩溃的时候丢失一秒钟的事务。为了保证事务的持久性和复制设置的一致性，建议将这个参数设置为 1。

8.2.6 存储和 I/O 相关参数的设置

数据库数据存储和 I/O 的性能直接影响到数据库数据文件读写执行的效率，合理地配置存储和 I/O 方面的参数，将会提升数据库整体的性能。

1. `innodb_open_files` 的设置

在 `mysqld` 中，对 `innodb_open_files` 参数的定义如下：

```
[root@localhost ~]# mysqld --verbose --help|grep "\-\\-innodb_open_files" -A 2
--innodb_open_files=#
```

How many files at the maximum InnoDB keeps open at the same time.

本参数的作用是限制 InnoDB 存储引擎在同一时间同时能打开表的数量，该参数默认值为 300，如果数据库里面的表特别多，可以考虑增加该参数的值。

2. innodb_flush_method 的设置

在 mysqld 中，对 innodb_flush_method 参数的定义如下：

```
[root@localhost ~]# mysqld --verbose --help|grep "\-\\-innodb_flush_method" -A 1
--innodb_flush_method=name
    With which method to flush data.
```

该参数的作用是设置 InnoDB 引擎与操作系统进行 I/O 交互的模式，即刷新数据和日志的方法，该参数有三个可选项，含义如下。

- Fdatasync (默认值): InnoDB 使用 fsync() 函数去更新日志和数据文件。
- O_DSYNC: InnoDB 使用 O_SYNC 模式打开并更新日志文件，使用 fsync() 函数去更新数据文件。
- O_DIRECT: InnoDB 使用 O_DIRECT 模式打开数据文件，使用 fsync() 函数去更新日志和数据文件。

3. innodb_max_dirty_pages_pct 的设置

在 mysqld 中，对 innodb_max_dirty_pages_pct 参数的定义如下：

```
[root@localhost ~]# mysqld --verbose --help|grep
"\-\\-innodb_max_dirty_pages_pct" -A 1
--innodb_max_dirty_pages_pct=#
    Percentage of dirty pages allowed in bufferpool.
```

该参数的作用是控制 InnoDB 的脏页在缓冲中的百分比，将脏页的比例控制在所设定的百分比值之下，如果该参数的值是 50，则脏页在缓冲中最多占 50%。建议该参数设置为 15~90，如果设置太大，缓存中每次更新需要置换的数据页太多；如果设置过小，可以存放脏数据页的缓冲区内内存空间会很小，性能会受到一定的影响。

8.2.7 其他重要参数的设置

除了上面介绍的参数外，还有一些其他的参数，也会影响服务器的性能。

1. max_connect_errors 的设置

在 mysqld 中，对 max_connect_errors 参数的定义如下：

```
max_connect_errors=10
```


`max_connect_errors` 默认值为 10，表示 `mysqld` 线程没重新启动过，一台物理服务器只要连接异常中断累计超过 10 次，就再也无法连接上 `mysqld` 服务。建议大家设置此值至少大小等于 10。若异常中断累计超过参数设置的值，有两种解决方法：可以重启 `mysqld` 服务或者执行 `FLUSH HOSTS` 命令。

2. interactive_timeout 的设置

在 `mysqld` 中，对 `interactive_timeout` 参数的定义如下：

```
[root@localhost ~]# mysql --verbose --help|grep '\-interactive_timeout' -A 2
--interactive_timeout=#
                        The number of seconds the server waits for activity on an
                        interactive connection before closing it.
```

该参数用于设置处于交互状态连接的活动被服务器端强制关闭后等待的时间。

3. wait_timeout

在 `mysqld` 中，对 `wait_timeout` 参数的定义如下：

```
[root@localhost ~]# mysql --verbose --help|grep '\-wait_timeout' -A 2
--wait_timeout=#      The number of seconds the server waits for activity on a
                        connection before closing it.
```

该参数用于设置客户端与服务器在无交互状态连接到被服务器端强制关闭而等待的时间。此参数只有针对基于 TCP/IP 或基于 Socket 通信协议建立的连接才有效。

4. query_cache_type

在 `mysqld` 中，对 `query_cache_type` 参数的定义如下：

```
[root@localhost ~]# mysql --verbose --help|grep '\-query_cache_type' -A 3
--query_cache_type=#
                        0 = OFF = Don't cache or retrieve results. 1 = ON = Cache
                        all results except SELECT SQL_NO_CACHE ... queries. 2 =
                        DEMAND = Cache only SELECT SQL_CACHE ... queries.
```

该参数用于控制查询结果是否放到查询缓存中。查询缓存类型的可选值只能是 0、1、2，具体选项的含义如下。

- 该参数选项设置为 0，表示禁止查询缓存的功能。
- 该参数选项设置为 1，表示启用查询缓存的功能，缓存所有符合要求的查询结果集，除 `SELECT SQL_NO_CACHE...`，以及不符合查询缓存设置的结果集外。
- 该参数选项设置为 2，表示仅仅缓存 `SELECT SQL_CACHE ...` 子句的查询结果集，除不符合查询缓存设置的结果集外。

5. query_cache_size

在 `mysqld` 中, 对 `query_cache_size` 参数的定义如下:

```
[root@localhost ~]# mysql --verbose --help|grep '\-query_cache_size' -A 1
--query_cache_size=#
                        The memory allocated to store results from old queries.
```

该参数用来设置查询缓存的大小。需要从以下几个方面考虑如何设置该参数的大小。

- 查询缓存区对 DDL 和 DML 语句的性能影响。
- 查询缓存区的内部维护成本。
- 查询缓存区的命中率以及内存使用率等因素。

8.3 MySQL 日志设置优化

MySQL 安装之后, 可以通过对其各个参数的修改来进行优化数据库服务的性能。数据库日志对 MySQL 的 I/O 性能有很大的影响, 本节主要介绍通过对日志参数的设置来提升数据库性能的方法。

下面先来看下二进制日志文件相关的参数, 如下所示。

```
mysql> show variables like '%binlog%';
```

| Variable_name | Value |
|-----------------------------------------|----------------------|
| binlog_cache_size | 32768 |
| binlog_direct_non_transactional_updates | OFF |
| binlog_format | MIXED |
| binlog_stmt_cache_size | 32768 |
| innodb_locks_unsafe_for_binlog | OFF |
| max_binlog_cache_size | 18446744073709547520 |
| max_binlog_size | 1073741824 |
| max_binlog_stmt_cache_size | 18446744073709547520 |
| sync_binlog | 0 |

9 rows in set (0.05 sec)

其中 `binlog_cache_size` 默认大小是 32768B, 即 32KB。该参数表示在事务中允许二进制日志缓存的大小。如果要设置该参数, 首先需要了解下 `binlog_cache_size` 的使用情况, 如下所示。

```
mysql> show status like '%binlog%';
```

| Variable_name | Value |
|---------------|-------|
|---------------|-------|

```

| Binlog_cache_disk_use      | 0      |
| Binlog_cache_use          | 0      |
| Binlog_stmt_cache_disk_use| 0      |
| Binlog_stmt_cache_use     | 0      |
| Com_binlog                 | 0      |
| Com_show_binlog_events    | 0      |
| Com_show_binlogs          | 0      |
+-----+-----+
7 rows in set (0.00 sec)

```

其中影响 binlog_cache_size 性能的参数主要如下：

- Binlog_cache_use: 表示使用二进制日志缓存事务的数量。
- Binlog_cache_disk_use: 该参数表示使用二进制日志缓存并且值达到 binlog_cache_size 设置的值，用临时文件存储事务的数量。

max_binlog_cache_size 参数表示二进制日志所使用缓存的最大值，该参数的默认值是 18446744073709547520，该默认值已经足够大了。

Max_binlog_size 参数代表二进制日志使用的最大值，如果系统中事务过多，而此参数数值设置过小，则会报错，该参数默认值是 1073741824(1G)，该参数一般设置为 512M 或者 1G。

Sync_binlog 参数比较重要，不仅对数据完整性有影响，而且对数据库的性能也有影响，该参数的选项如下所示：

- Sync_binlog=0: 表示当事务提交之后，不做文件系统之类的磁盘同步指令刷新 binlog_cache 中的信息到磁盘，而会让文件系统自行决定同步，或者 cache 满了之后才同步到磁盘。
- Sync_binlog=n: 表示当事务提交 n 次之后，将进行一次同步，将 binlog_cache 中的数据写入到磁盘。

值得注意的是，Sync_binlog=0 这是数据库默认的设置，性能是最好的，不过也是最危险的，一旦系统崩溃，binlog_cache 中的数据就会丢失。Sync_binlog=1 的时候，是数据库消耗最大，不过安全性比较高，当事务提交后，数据库会将缓存中的数据同步到磁盘中。

通常在 MySQL 的复制过程中，实际上就是将客户端的日志通过 I/O 复制到服务器端，然后服务器端将日志中的数据解析出来应用到数据库中。

8.4 MySQL I/O 设置优化

目前大多数应用都是以 I/O 密集型为主，存储技术远没有计算机中其他系统发展迅速，尽管也有不少高端存储设备，不过考虑到成本因素，一般用户承受不起，目前大多数采用 SAS

盘结合应用不同的 RAID 组合，来实现存储。

事实上，前面提到的 SQL 优化、数据库对象优化、数据库参数优化，以及应用程序优化等，都是想通过减少或延缓磁盘的读写来减轻磁盘 I/O 的压力。另外，管理员可以通过增强磁盘 I/O 的吞吐量以及增强磁盘 I/O 本身的性能，从而提高数据库的整体性能。

下面针对磁盘 I/O 进行不同类型的优化。

1. 选择合适的 RAID（磁盘阵列）级别

RAID 是 Redundant Array of Independent Disk 的缩写，就是将 N 台硬盘通过 RAID Controller 结合成虚拟单台大容量硬盘使用。根据数据冗余和分布方式，RAID 分为不同的级别，下面是最常见的几种 RAID 级别，如表 8-1 所示。

表 8-1 常见 RAID 级别的比较

| RAID 级别 | 特性 | 优点 | 缺点 |
|---------|--------------------------------------------------------------------------------|----------------------------------------|-------------------------------------------------------|
| RAID 0 | 也叫条带化（Stripe），按一定的条带大小将数据依次分布到各个磁盘，没有数据冗余 | 数据并发读写速度快，无额外磁盘空间开销 | 数据无冗余保护，可靠性差 |
| RAID 1 | 也叫磁盘镜像（Mirror），两个磁盘一组，所有数据都同时写入两个磁盘，但读时从任一磁盘读都可以 | 可靠性高，并发读写性能优良 | 容量一定的话，需要 2 倍的磁盘，投资大 |
| RAID 10 | 也叫 RAID1+0，是 RAID 0 与 RAID 1 的组合物，集成了 RAID0 快速和 RAID 1 的安全 | 可靠性能高，并发读写性能优良 | RAID10 会造成 50% 的磁盘的浪费。投资比较大 |
| RAID 3 | RAID3 是把数据分成多个块，按照一定的容错算法，存放在 N+1 个硬盘上，事实上空间是 N 个磁盘的空间总和，第 N+1 个磁盘存储的信息是校验容错信息 | 当 N+1 个硬盘中出现故障时，从其他 N 个硬盘中恢复原始数据 | RAID 3 校验磁盘很容易成为整个系统的瓶颈。校验盘的负载如果很大，会导致整个 RAID 系统性能的下降 |
| RAID 4 | 跟 RAID0 一样对磁盘组条带化，不同的是：需要额外增加一个磁盘，用来写各校验纠错数据 | RAID 中一个磁盘损坏，其他数据可以通过校验纠错数据计算出来，读数据速度快 | 在失败恢复的时候，难度可要比 RAID3 大，控制器的设计难度也要大许多，而且访问数据的效率不高 |
| RAID 5 | RAID5 是一种存储性能、数据安全和存储成本兼顾的存储解决方案，将每个条带的校验纠错数据块也分布到各个磁盘，而不是写到一个磁盘 | 跟 RAID 4 相似，只是其写性能和数据保护能力要更强一点 | 容错能力不及 RAID 1，在磁盘出现损坏时，读写能力会下降 |

了解了 RAID 级别的特性后，就可以根据数据读写的特点、可靠性要求，以及投资额度来选择合适的 RAID 级别了，比如：

- 数据读写都比较频繁，最好选择 RAID 10；
- 数据读很频繁，写相对少一点，对可靠性有一定的要求，可以考虑选择 RAID 5；

- 数据读写比较频繁，但是对可靠性能要求不高，可以选择 RAID10。

2. 使用 Symbolic Links 分布 I/O

使用操作系统的符号链接（Symbolic Links）将不同的数据库或表、索引指向不同的物理磁盘，从而达到分布磁盘 I/O 的目的。可以通过 `SHOW VARIABLES LIKE 'have_symlink'` 语句，检查系统是否支持符号链接。

```
mysql> SHOW VARIABLES LIKE 'have_symlink';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| have_symlink  | YES   |
+-----+-----+
1 row in set (0.00 sec)
```

在不使用 RAID 或者逻辑卷的情况下，要想达到分布式磁盘 I/O 的目的，可以使用操作系统的 Symbolic Links 将不同的数据库或表、索引执行在不同的物理磁盘上面。

(1) 将一个数据库指向其他物理磁盘，首先在目标磁盘上创建目录，然后再创建从 MySQL 数据目录到目标目录的符号链接，命令如下所示。

```
[root@localhost ~]# mkdir /otherdisk/databases/test
[root@localhost ~]# ln -s /otherdisk/databases/test /path/to/datadir
```

(2) 对于新建的 MyISAM 表，可以使用 DATA DIRECTORY 和 INDEX DIRECTORY 来将表的数据文件或索引文件指向其他物理磁盘，命令如下所示。

```
Create table test(id int primary key,name varchar(20))
Type = myisam
DATA DIRECTORY = '/disk2/data'
INDEX DIRECTORY = '/disk3/index'
```

提示

对于已经存在的 MyISAM 表，可以将 .MYD 或者 .MYI 文件移到目标磁盘，然后再建立符号链接即可。

8.5 MySQL 并发设置优化

MySQL 数据库并发性能是影响数据整体性能的一个重要的参数，通过提高并发连接数量来提升并发效率，下面来了解下如何优化数据库连接的最大数。

首先查看一下并发连接的最大数和目前连接的最大数，命令如下：

```
mysql> show variables like 'max_connections';
+-----+-----+
```

```
| Variable_name | Value |
+-----+-----+
| max_connections | 500 |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> show global status like 'max_used_connections';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Max_used_connections | 125 |
+-----+-----+
1 row in set (0.00 sec)
```

参数 `max_connections` 表示允许同时连接 MySQL 数据库客户端的最大数量，`max_used_connections` 表示同时使用的最大连接数。`max_used_connections` 占 `max_connections` 的 80% 左右比较合适。一般来讲 `max_used_connections` 参数在 MySQL 主机性能允许的范围内，可以考虑设置 500~800 比较合适，而参数 `max_connections` 是用户正在连接最大的数量，可以尽量设置大一些。

参数 `net_buffer_length` 设置的只是消息缓冲区的初始化大小，系统默认值是 16KB。当然，如果系统内存比较紧张的情况下，可以考虑将该参数降低到 8KB 左右比较合适。查看命令如下：

```
mysql> show variables like 'net%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| net_buffer_length | 16384 |
| net_read_timeout | 30 |
| net_retry_count | 10 |
| net_write_timeout | 60 |
+-----+-----+
4 rows in set (0.00 sec)
```

参数 `max_allowed_packet` 表示在网络传输中，一次传递数据量的最大值。系统默认值为 1MB，最大值是 1GB，必须将该参数设置为 1024 的倍数，单位为字节。查看命令如下：

```
mysql> show variables like 'max_allowed_packet%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_allowed_packet | 1048576 |
+-----+-----+
1 row in set (0.00 sec)
```


参数 `back_log` 表示在 MySQL 的连接请求等待队列中允许存放的最大连接请求数。系统默认的值是 50，最大可以设置为 65535。查看命令如下：

```
mysql> show variables like 'back_log%';
```

| Variable_name | Value |
|---------------|-------|
| back_log | 50 |

1 row in set (0.00 sec)

8.6 线程、Table Cache 和临时表的优化

除了上面章节介绍的优化方法以外，还有一些因素也能优化 MySQL 的整体性能。例如：线程管理，Table Cache 管理和临时表管理。

8.6.1 线程的优化

线程的优化主要是优化 MySQL 中与连接线程相关的系统参数以及状态变量。在 MySQL 中实现了一个线程池，将空闲的连接线程存放在线程池中，如果有新的请求，MySQL 会检查线程池中是否有空闲的连接线程，如果没有会创建新的线程，如果线程池中有空闲的线程，会直接从线程池中取出空闲的线程。

查看与连接线程相关的系统参数，如下所示。

```
mysql> show variables like 'thread%';
```

| Variable_name | Value |
|--------------------|---------------------------|
| thread_cache_size | 64 |
| thread_concurrency | 10 |
| thread_handling | one-thread-per-connection |
| thread_stack | 196608 |

4 rows in set (0.04 sec)

参数 `thread_cache_size` 表示数据库线程池中应该存放的连接线程数。`thread_cache_size` 默认配置 `thread_cache_size=8`。建议根据物理内存设置，如果 1GB 内存可以设置为 8，2GB 的内存可以设置为 16，3GB 内存可以考虑设置 32，大于 3GB 的内存可以设置成 64。

参数 `thread_stack` 表示每个连接线程被创建的时候，MySQL 分配内存的大小。系统默认值是 192KB，MySQL 的 `max_connections * thread_stack` 应小于可用内存。

8.6.2 关于 table_cache 相关的优化

table_cache 是非常重要的 MySQL 性能参数，主要用于设置 table 高速缓存的数量。由于每个客户端连接都会至少访问一个表，所以此参数的值与 max_connections 有关。

当某一连接访问一个表时，MySQL 会检查当前已缓存表的数量。如果该表已经在缓存中打开，则会直接访问缓存中的表已加快查询速度；如果该表未被缓存，则会将当前的表添加进缓存并进行查询。

在执行缓存操作之前，table_cache 用于限制缓存表的最大数目。如果当前已经缓存的表未达到 table_cache 的值，则会将新表添加进来；若已经达到此值，MySQL 将根据缓存表的最后查询时间、查询率等规则释放之前的缓存。

查看 table_cache 相关的系统参数，如下所示。

```
mysql> show variables like 'table_cache';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| table_cache   | 512   |
+-----+-----+
1 row in set (0.06 sec)
```

上述结果显示系统设置的 table_cache 为 512 个。

通过 open_tables 可以查看当前系统中已打开的表的数目，如下所示。

```
mysql> show status like 'open_tables';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Open_tables   | 6     |
+-----+-----+
1 row in set (0.00 sec)
```

上述结果显示已经打开的表为 6 个。

table_cache 只是表高速缓存的大小。用户访问 MySQL 数据库中的某张表时，如果 MySQL 的缓冲区还存在空间，那么会将被打开的表放入到缓冲区中，这样做的好处是可以更快速地访问表中内容。

在 MySQL 默认的情况下，对于内存存在 2GB 以下的机器，table_cache 的默认值是 256 到 512，如果内存有 4GB，则默认值是 2048，所以说机器内存越大，table_cache 的值越大，对于 table_cache 加大后，MySQL 的执行速度更快了，同时产生死锁的几率也会增加，所以在平时维护过程中，应该找到合适的值，从而在 MySQL 的性能和产生死锁的几率中需找平衡。

对于 table_cache 加大之后有可能存在文件描述符不够用的情况，MySQL 的配置文件中有这么一段提示信息，如下所示。

```
"The number of open tables for all threads. Increasing this value increases the
```

```
number of file descriptors that mysqld requires. Therefore you have to make sure
to set the amount of open files allowed to at least 4096 in the variable
"open-files-limit" in" section [mysqld_safe]"
```

由此可见, `table_cache` 的值加大后有可能存在文件描述符不够的情况, 设置参数 `open-files-limit` 用来限制系统打开文件的最大值, 解释如下。

```
[root@localhost bin]# mysqld --verbose --help|grep 'open-files-limit' -A 6
--open-files-limit=#
        If this is not 0, then mysqld will use this value to
        reserve file descriptors to use with setrlimit(). If this
        value is 0 then mysqld will reserve max_connections*5 or
        max_connections + table_cache*2 (whichever is larger)
        number of file descriptors
open-files-limit                                2500
```

很多朋友可能遇到文件描述符不够的情况, 可以采用如下方法解决。

```
[root@localhost ~]# ulimit -n 5000
```

这种方式的调整只是临时性的, 如果换个中断连接, 该值又会回到原始值, 可以考虑通过 `vi` 将命令 `ulimit -n 5000` 写入到 `/etc/profile`, 服务器重启就可以生效, 同时还建议把 `/etc/security/limits.conf` 也修改下, 还要在配置文件中把 `open_files_limit` 这个参数增大, 对于 4GB 内存的数据库服务器, 可以把 `open_files_limit` 至少要增大到 4096, 如果没有什么特殊情况, 设置成 8192 就可以了。

8.6.3 关于临时表的优化

临时文件就是为了各种不同的目的, 而产生的中间文件。使用完毕后会及时地回收和清理。临时表也是如此, 是 MySQL 在进行一些内部操作的时候生成的数据库表。这些操作主要包括 `group by`、`distinct`、一些 `order by` 查询语句、`UNION` 和一些 `from` 语句中的子查询等。

可以使用 `EXPLAIN` 来分析查询语句, 看看是否会用到临时表。`EXPLAIN` 输出中的 `EXTRA` 列会指明是否 “Using temporary”。事实上, 大多数用户都不会去关注临时表的产生、使用与消亡的过程, 因为它对用户是透明的。但是对于数据库管理员或者其他关心性能的人员而言, 临时表就不得不引起注意, 因为如果设置不当或者是程序使用不当, 可能会产生大量的磁盘临时表, 对系统性能产生很大的影响。

什么是磁盘临时表呢? 除了直接产生的磁盘临时表外, 大量磁盘临时表是由内存临时表转化来的。临时表是存在于内存中, 由 `MEMORY` 引擎进行处理, 速度较快。而磁盘临时表则是在磁盘上创建、使用、销毁的。由于磁盘是慢速访问设备, 因此, 磁盘临时表的操作效率要比临时表的操作差了几个数量级, 具有较差的性能。因此, 需要尽量避免磁盘临时表的产生。

查看临时文件、临时表和磁盘临时表的命令如下:

```
mysql> show global status like 'created_tmp%';
+-----+-----+
```



```

| Variable_name | Value |
+-----+-----+
| Created_tmp_disk_tables | 0 |
| Created_tmp_files | 5 |
| Created_tmp_tables | 2 |
+-----+-----+
3 rows in set (0.00 sec)

```

数据库系统每次创建临时表时，Created_tmp_tables 会增加，如果是在磁盘上创建临时表，Created_tmp_disk_tables 也会增加，Created_tmp_files 表示 MySQL 数据库创建的临时文件的数量。

下面看下 MySQL 数据库对临时表的配置，如下所示。

```

mysql> show variables where
Variable_name in('max_heap_table_size','tmp_table_size');
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_heap_table_size | 16777216 |
| tmp_table_size | 536870912 |
+-----+-----+
2 rows in set (0.00 sec)

```

max_heap_table_size 参数表示 heap 数据表的最大长度，默认设置是 16MB。值得注意的是有 text 或者 blob 字段时，会导致不能用内存临时表而用磁盘临时表。

tmp_table_size 参数规定了内存临时表最大值（推荐该参数设置 200MB），如果内存临时表超过了限制，数据库会将数据存储于磁盘临时表中，存储的目录默认是/tmp/，如下所示。

```

mysql> show variables like 'tmpdir';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| tmpdir | /tmp |
+-----+-----+
1 row in set (0.01 sec)

```

8.7 小结

本章主要讲解了 MySQL 服务器优化方面的一些知识和技巧，涉及 MySQL 服务器安装优化，MySQL 服务器配置优化，以及如何通过其他优化达到提升 MySQL 性能。其中，MySQL 的 I/O 性能是制约 MySQL 性能的一个非常重要的方面。通过本章的学习，可以全面地提升 MySQL 的整体性能。通过参数设置进行数据库性能优化所带来的性能提升很有限，在调优的过程中，除了依赖于数据库参数配置提升整体性能，同时也应该考虑应用设计和程序设计过程中尽可能减少存在影响数据库性能的因素。

第 9 章

◀ MySQL 性能监控 ▶

在 MySQL 数据库中，通常使用第三方软件来监控数据库性能健康状况。本章重点介绍了监控利器 Nagios 工具。使用 Nagios 监控工具不仅可以监控 MySQL 数据库性能，还可以对数据库集群、数据复制等操作进行监控。因为数据库性能的瓶颈不仅仅和数据库本身的优化情况有关，还和操作系统本身的情况，以及网络负载的状况有关。因此，本章还介绍了如何监控 Linux 操作系统的 I/O 和 CPU 的运行状况。通过本章的学习可以更好地对 MySQL 数据库以及服务器的性能进行全方位的监控。

9.1 基本监控系统方法

在 Linux 操作系统中，用户可以使用一些分析系统性能的命令去分析数据库服务器的性能。本章节主要介绍这些命令的使用方法。

9.1.1 ps 命令

ps 命令主要用来获取对于某个进程的一些信息。ps 命令的常用参数如下所示。

- -A: 显示所有进程。
- -a: 显示一个终端的所有进程，除了会话引线。
- -N: 忽略选择。
- -d: 显示所有进程，但省略所有的会话引线。
- -e: 列出程序时，显示每个程序所使用的环境变量。
- -x: 显示没有控制终端的进程，同时显示各个命令的具体路径。dx 不可合用。
- -p pid: 进程使用 cpu 的时间。
- -u uid or username: 选择有效的用户 id 或者是用户名。
- -g gid or groupname: 显示组的所有进程。
- -f: 全部列出，通常和其他选项联用。
- -l: 长格式（有 F, wchan, C 等字段）。
- -j: 作业格式。
- -e: 命令之后显示环境。

例如，运行一下 `ps -aux` 命令显示所有进程的信息，如下所示。

```
[root@localhost ~]# ps -aux
Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.7/FAQ
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  1948   740 ?        Ss   Nov19    0:03 /sbin/init
root         2  0.0  0.0      0     0 ?        S<   Nov19    0:00 [kthreadd]
...
nagios  20727  0.0  0.1  12952  1012 ?        Ssl  11:12    0:03
/var/www/nagios/bin/nagios -d /
nagios  21037  0.0  0.1   4960   908 ?        Ss   11:23    0:00 ./nrpe -c
/var/www/nagios/etc/n
```

可以使用 `ps -ef` 命令查询所有进程及其环境变量信息，如下所示。

```
[root@localhost ~]# ps -ef
UID      PID PPID  C  STIME TTY          TIME CMD
root         1    0  0 Nov19 ?           00:00:03 /sbin/init
root         2    0  0 Nov19 ?           00:00:00 [kthreadd]
...
root    17334     1  0 Nov20 ?           00:00:00 /bin/sh
/usr/local/mysql/bin/mysqld_safe --datadir
nagios  21037     1  0 11:23 ?           00:00:00 ./nrpe -c
/var/www/nagios/etc/nrpe.cfg -d
nagios  23516 23515  0 14:52 ?           00:00:00
/var/www/nagios/libexec/check_ssh 127.0.0.1,192.16
root    23524 20118  3 14:52 pts/3      00:00:00 ps -ef
```

通常查看了进程信息后，如果需要终止某个进程，可以使用 `kill` 命令，如下。

```
[root@localhost ~]# kill -KILL [pid]
```

如果需要强行终止某个进程的话，可以使用 `kill -9 [pid]` 命令，如下。

```
[root@localhost ~]# kill -9 [pid]
```

9.1.2 top 命令

`top` 命令是 Linux 操作系统下最常用的性能分析工具，能够实时地显示操作系统中各个进程消耗资源的情况。该命令可以显示 CPU 使用、内存使用和执行时间，下面执行 `top` 命令看下执行的情况，如下所示。

```
[root@localhost ~]# top
top - 15:12:58 up 3 days, 4:13, 4 users, load average: 0.29, 0.27, 0.28
Tasks: 138 total, 3 running, 133 sleeping, 1 stopped, 1 zombie
Cpu(s): 24.4%us, 9.2%sy, 0.0%ni, 66.4%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 673164k total, 650156k used, 23008k free, 90248k buffers
Swap: 524280k total, 52k used, 524228k free, 185692k cached
```


| PID | USER | PR | NI | VIRT | RES | SHR | S | %CPU | %MEM | TIME+ | COMMAND |
|-------|------|----|----|-------|-----|------|---|------|------|----------|----------------|
| 2464 | root | 20 | 0 | 47108 | 20m | 7888 | R | 15.9 | 3.1 | 39:50.58 | Xorg |
| 20115 | root | 20 | 0 | 90728 | 18m | 11m | S | 4.0 | 2.8 | 0:07.86 | gnome-terminal |
| ... | | | | | | | | | | | |
| 62 | root | 15 | -5 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | kacpi_notify |
| 120 | root | 15 | -5 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.00 | cqueue |
| 122 | root | 15 | -5 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:00.04 | ksuspend_usbd |
| 127 | root | 15 | -5 | 0 | 0 | 0 | S | 0.0 | 0.0 | 0:01.09 | khubd |

下面分析一下 top 命令的统计信息的含义。

```
top - 15:12:58 up 3 days, 4:13, 4 users, load average: 0.29, 0.27, 0.28
```

先来分析 top 命令第 1 行信息，该行信息的具体含义如下。

- 5:12:58 表示系统运行的当前时间。
- up 3 days 表示系统运行时间。
- 4 users 表示登录用户的数量。
- load average: 0.29, 0.27, 0.28 表示系统负载，即任务队列的平均长度，三个数值分别为 1 分钟、5 分钟、15 分钟到现在的平均值。

接着分析 top 命令的第 2 行和第 3 行的具体含义，这两行分别表示进程和 CPU 的性能的一些信息。

```
Tasks: 138 total, 3 running, 133 sleeping, 1 stopped, 1 zombie
Cpu(s): 24.4%us, 9.2%sy, 0.0%ni, 66.4%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
```

下面我们具体分析一下各个统计信息的含义，如下所示。

- 138 total 表示进程的总数。
- 3 running 表示正在运行的进程数量。
- 133 sleeping 表示睡眠的进程数量。
- 1 stopped 表示停止的进程数量。
- 1 zombie 表示僵尸进程数。
- 24.4%us 表示用户空间占用 CPU 的百分比。
- 9.2%sy 表示内核空间占用 CPU 的百分比。
- 0.0%ni 表示用户进程空间内改变过优先级的进程占用 CPU 的百分比。
- 66.4%id 表示空闲 CPU 的百分比。
- 0.0%wa 表示等待输入输出的 CPU 的百分比。

接着分析 top 命令的第 4 行和第 5 行的具体含义，这两行分别表示内存的性能分析信息，如下。

```
Mem: 673164k total, 650156k used, 23008k free, 90248k buffers
```

下面具体分析一下 Mem 各个统计信息的含义，如下所示。

- 673164k total 表示物理内存的总的大小。
- 650156k used 表示使用的物理内存的大小。
- 23008k free 表示空闲的物理内存的大小。
- 90248k buffers 表示内核缓存内存空间的大小。

```
Swap: 524280k total, 52k used, 524228k free, 185692k cached
```

下面具体分析一下 Swap 各个统计信息的含义，如下所示。

- 524280k total 表示交换区的总量。
- 52k used 表示使用的交换区总量。
- 524228k free 表示空闲交换区总量。
- 185692k cached 表示缓冲的交换区总量。内存中的内容被换出到交换区，而后又被换入到内存，但使用过的交换区尚未被覆盖，该数值即为这些内容已存在于内存中的交换区的大小。相应的内存再次被换出时可不必再对交换区写入。

9.1.3 vmstat 命令

vmstat 命令可以用来显示 Linux 性能指标，该命令分别输出进程、内存、交互区、I/O、系统和 CPU 的情况。下面直接输出该命令，如下所示。

```
[root@localhost ~]# vmstat
procs -----memory----- --swap-- -----io----- --system--
-----cpu-----
r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
2  0    404 12264 101280 164840  0  0   3  33   6 113  4  2 94  0  0
```

首先，分析下进程(procs)的两列信息，如下所示。

- r 列表示可运行进程的数量。
- b 列表示阻塞进程的数量。

内存性能有 4 个报告虚拟内存如何使用的字段，具体意义如下。

- swpd 表示已经使用的交换空间的数量。
- free 表示自由 RAM 数量。
- buff 表示缓存使用的 RAM 的数量。
- cache 表示文件系统缓存使用的 RAM 数量。

swap 交换字段进行详细说明，如下所示。

- si 表示从磁盘分页到内存的数量。
- so 表示从内存分页到磁盘的数量。

io 字段进行详细说明，如下所示。

- bi 表示从磁盘读入的块。
- bo 表示写入磁盘的块。

下面，对系统字段和 CPU 字段进行说明，CPU 状态使用总 CPU 时间的百分比来表示，具体含义如下。

- in 表示系统中断。
- cs 表示进程上下文开关。
- us 表示用户模式。
- sy 表示内核模式。
- wa 表示等待 I/O。
- id 表示空闲状态。

9.1.4 mytop 命令

mytop 是一个类似于 Linux 下的 top 命令风格的 MySQL 监控工具，可以监控当前用户正在执行的命令。

首先，使用源码安装 mytop-1.6.tar.gz，如下所示。

```
[root@localhost ~]# tar -zxvf mytop-1.6.tar.gz
[root@localhost ~]# cd mytop-1.6
[root@localhost mytop-1.6]# perl Makefile.PL
[root@localhost mytop-1.6]# make
[root@localhost mytop-1.6]# make test
[root@localhost mytop-1.6]# make install
[root@localhost mytop-1.6]#
```

直接输入 mytop 命令，结果发生错误，如下所示。

```
[root@localhost mytop-1.6]# mytop
Can't locate DBI.pm in @INC (@INC contains:
/usr/lib/perl5/5.10.0/i386-linux-thread-multi /usr/lib/perl5/5.10.0
/usr/lib/perl5/site_perl/5.10.0/i386-linux-thread-multi
/usr/lib/perl5/site_perl/5.10.0 /usr/lib/perl5/site_perl/5.8.8
/usr/lib/perl5/site_perl/5.8.7 /usr/lib/perl5/site_perl/5.8.6
/usr/lib/perl5/site_perl/5.8.5 /usr/lib/perl5/site_perl
/usr/lib/perl5/vendor_perl/5.10.0/i386-linux-thread-multi
/usr/lib/perl5/vendor_perl/5.10.0 /usr/lib/perl5/vendor_perl/5.8.8
/usr/lib/perl5/vendor_perl/5.8.7 /usr/lib/perl5/vendor_perl/5.8.6
/usr/lib/perl5/vendor_perl/5.8.5 /usr/lib/perl5/vendor_perl .) at /usr/bin/mytop
line 22.

BEGIN failed--compilation aborted at /usr/bin/mytop line 22.
```


发现报错了，Can't locate DBI.pm in @INC 这种问题主要原因是系统没有安装 DBI 组件，可以安装 DBI、Data-ShowTable、DBD-mysql（假设已安装完 perl 和 mysql 数据库）。

下面可以直接到 <ftp://ftp.funet.fi/pub/languages/perl/CPAN/modules/by-module> 下载三个文件 DBI-1.601.tar.gz、Data-ShowTable-3.3.tar.gz、DBD-mysql-3.0007_1.tar.gz，分别处于 DBI、Data、DBD 目录下。

首先，先解压这三个压缩文件，如下所示。

```
[root@localhost ~]# tar zxvf DBI-1.601.tar.gz
[root@localhost ~]# tar zxvf Data-ShowTable-3.3.tar.gz
[root@localhost ~]# tar zxvf DBD-mysql-3.0007.tar.gz
```

下面安装 DBI，如下所示。

```
[root@localhost ~]# cd DBI-1.601
[root@localhost DBI-1.601]# perl ./Makefile.PL
[root@localhost DBI-1.601]# make
[root@localhost DBI-1.601]# make test
[root@localhost DBI-1.601]# make install
```

安装 Data-ShowTable，如下所示。

```
[root@localhost ~]# cd Data-ShowTable-3.3
[root@localhost Data-ShowTable-3.3]# perl ./Makefile.PL
[root@localhost Data-ShowTable-3.3]# make
[root@localhost Data-ShowTable-3.3]# make install （注：无需 make test）
```

安装 DBD-mysql，如下所示。

```
[root@localhost src]# cd DBD-mysql-3.0007_1
[root@localhost DBD-mysql-3.0007_1]# perl ./Makefile.PL
--libs="-L/usr/local/mysql-6.0.9-alpha/lib/mysql -lmysqlclient -lz -lrt -lcrypt
-lns1 -lm" --cflags="-I/usr/local/mysql-6.0.9-alpha/include/mysql -g
-DUNIX_LINUX" --testuser=root --testsocket=/home/cserken/mysql/tmp/mysql.sock
[root@localhost DBD-mysql-3.0007_1]# make
[root@localhost DBD-mysql-3.0007_1]# make test
[root@localhost DBD-mysql-3.0007_1]# make install
```

测试一下，结果问题还没有解决，发现没有安装 mytop 所需 TermReadKey，如下所示。

```
[root@localhost ~]# tar -xzvf TermReadKey-2.30.tar.gz
[root@localhost ~]# cd TermReadKey-2.30
[root@localhost TermReadKey-2.30]# perl ./Makefile.PL
[root@localhost TermReadKey-2.30]# make
[root@localhost TermReadKey-2.30]# make install
```

下面测试一下 mytop 命令，结果显示正常运行了，如下所示。

```
[root@localhost ~]# cd mytop-1.6
```

```
[root@localhost mytop-1.6]# mytop
MySQL on localhost (5.5.27-ndb-7.2.8-cluster-gpl-log)   up 4+11:10:38
[19:18:03]
Queries: 3.0   qps:   0 Slow:   0.0           Se/In/Up/De(%):   00/00/00/00
[root@localhost ~]#
Key Efficiency: 75.0% Bps in/out:   0.0/   0.0
```

| Id | User | Host/IP | DB | Time | Cmd Query or State |
|------|------|-----------|------|------|-------------------------|
| 3835 | root | localhost | test | 0 | Query show full process |

下面先了解下 mytop 命令参数的含义:

- -u / --user <USERNAME> 指定 username, 预设是 root。
- -p / --pass / --password <PASSWORD>: 指定 password, 预设是 none。
- -h / --host <HOSTNAME[:PORT]>: 指定 MySQL server 的 hostname, 预设是 localhost。
- -P / --port <PORT>: 指定连接 MySQL server 的 port, 预设是 3306。
- -s / --delay <SECONDS>: 更新的秒数, 预设是 5 秒。
- -d / --db / --database <DATABASE>: 指定连接的资料库, 预设是 test。
- -b / --batch / --batchmode: 指定为 batch mode, 每次更新不会清除旧的显示结果, 会将更新资料显示在最上方, 预设是 unset。
- -S / --socket <PATH_TO_SOCKET>: 指定使用 MySQL socket 直接连线, 而不使用 TCP/IP 连线, 预设是 none (当 mytop 和 MySQL 在同一台时才能使用)。
- --header or --noheader: 是否要显示表头, 预设是 header。
- --color or --nocolor: 是否要使用颜色, 预设是 color。
- -i / --idle or --noidle: idle 的 thread 是否要出现在清单上, 预设是 idle。

先在 MySQL 数据库中创建 xfimti 用户, 如下所示。

```
mysql> grant all privileges on *.* to 'xfimti'@'localhost' identified by '123';
Query OK, 0 rows affected (0.01 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.02 sec)
```

下面输入 mytop 命令, 监控信息如下所示。

```
[root@localhost ~]# cd mytop-1.6
[root@localhost mytop-1.6]# ./mytop -u xfimti -p123 -d test -s
/temp/mysql.socket

MySQL on localhost (5.5.27-ndb-7.2.8-cluster-gpl-log)   up 4+11:39:16
[19:46:41]
```

Queries: 3.0 qps: 0 Slow: 0.0 Se/In/Up/De(%): 00/00/00/00

Key Efficiency: 83.3% Bps in/out: 0.0/ 0.0

| Id | User | Host/IP | DB | Time | Cmd Query or State |
|------|--------|-----------|------|------|-------------------------|
| --- | ---- | ----- | -- | ---- | ----- |
| 3875 | xfimti | localhost | test | 0 | Query show full process |

9.1.5 sysstat 工具

sysstat 工具包含检测系统性能及效率的一组工具。例如 CPU 的使用率、硬盘和网络吞吐数据，这些数据的收集和分析，有利于判断系统是否正常运行。如果是通过源码包安装，请到官方下载源码包，地址为 <http://perso.wanadoo.fr/sebastien.godard>。

Sysstat 软件包集成如下工具：

- iostat 工具：提供 CPU 使用率及硬盘吞吐效率的数据。
- mpstat 工具：提供单个处理器或多个处理器相关数据。
- sar 工具：负责收集、报告并存储系统活跃的信息。
- sa1 工具：负责收集并存储每天系统动态信息到一个二进制的文件中。
- sa2 工具：负责把每天的系统活跃信息写入总结性的报告中。
- sadc 工具：系统动态数据收集工具，收集的数据被写进一个二进制的文件中。
- sady 工具：显示被 sar 工具通过多种格式收集的数据。

下面开始安装 sysstat 工具包，如下所示。

```
[root@localhost ~]# tar zxvf sysstat-10.0.0.tar.gz
[root@localhost sysstat-10.0.0]# ./configure
[root@localhost sysstat-10.0.0]# make
[root@localhost sysstat-10.0.0]# make install
```

1. iostat 工具

Iostat 工具用于输出 CPU 和磁盘 I/O 相关的统计信息，具体的语法格式如下：

```
iostat [-c|-d] [-k] [-t] [-V] [-x] [-p device|ALL] [间隔描述] [检测次数]
```

上述参数的含义如下：

- -c：表示仅显示 CPU 的状态。
- -d：仅显示存储设备的状态，不可以和 -c 一起使用。
- -k：默认显示的是读入读出的块信息。
- -t：显示搜集数据的时间。
- -V：显示版本号和帮助信息。
- -x：显示扩展信息。

- -p device|ALL: device 为某个设备或者某个分区, 如果使用 ALL, 就表示要显示所有分区和设备的信息。

直接运行 Iostat 命令, 结果如下所示。

```
[root@localhost ~]# iostat
Linux 2.6.25-14.fc9.i686 (localhost.localdomain) 11/23/2012 _i686_ (1
CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.60    0.04   1.53    0.49    0.00   94.34

Device:            tps    kB_read/s    kB_wrtn/s    kB_read  kB_wrtn
sda                  1.66         2.47         32.28    800397   10474295
dm-0                  8.34         2.46         32.28    797729   10473872
dm-1                  0.00         0.00         0.00       592       404
sdb                   0.00         0.01         0.00     4433         1
```

结果中关于 CPU 性能参数的含义如下:

- %user: 在用户级别运行所使用的 CPU 的百分比。
- %nice: nice 操作所使用的 CPU 的百分比。
- %system: 在系统级别 (kernel) 运行所使用 CPU 的百分比。
- %iowait: CPU 等待硬件 I/O 时, 所占用 CPU 的百分比。
- %idle: 表示 CPU 空闲时间所占比例。

结果中关于磁盘 I/O 性能参数的含义如下:

- Device: 表示设备块的名字。
- tps: 表示每秒钟发送到的 I/O 请求数。
- kB_read/s: 表示从该设备每秒读取的数据块数量。
- kB_wrtn/s: 表示从该设备每秒写入的数据块数量。
- kB_read: 表示从该设备读取的数据块总数。
- kB_wrtn: 表示从该设备写入的数据块总数。

使用 -x 参数可以获得更多的统计信息, 如下所示。

```
[root@localhost init.d]# iostat -d -x -k 1 10
Linux 2.6.25-14.fc9.i686 (localhost.localdomain) 11/23/2012 _i686_ (1
CPU)

Device:            rrqm/s  wrqm/s    r/s    w/s    rkB/s    wkB/s avgrq-sz
avgqu-sz  await  r_await  w_await  svctm  %util
sda        0.03    6.65    0.25   1.42     2.47    32.28    41.81    0.16
98.24  28.60 110.40   6.41   1.07
dm-0        0.00    0.00    0.27   8.07     2.46    32.28    8.33    1.35
```

| | | | | | | | | | | |
|--------|-------|--------|------|------|------|------|------|-------|------|------|
| 161.51 | 34.13 | 165.77 | 1.28 | 1.06 | | | | | | |
| dm-1 | | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 8.00 | 0.00 |
| 53.71 | 52.81 | 55.04 | 7.02 | 0.00 | | | | | | |
| sdb | | 0.02 | 0.00 | 0.00 | 0.00 | 0.01 | 0.00 | 37.26 | 0.00 | |
| 16.76 | 15.00 | 225.50 | 9.53 | 0.00 | | | | | | |

结果中主要的参数的含义如下：

- rrqm/s: 表示每秒这个设备相关的读取请求有多少被合并。
- wrqm/s: 表示每秒这个设备相关的写入请求有多少被合并。
- r/s: 表示每秒请求读该设备的数量。
- w/s: 表示每秒请求写该设备的数量。
- await: 表示每一个 I/O 请求处理的平均时间。
- %util: 表示在统计时间内所有处理的 I/O 时间，除以总共统计的时间。

此外可以通过如下命令查询 CPU 的部分信息，如下所示。

```
[root@localhost ~]# iostat -c 1 10
avg-cpu:  %user   %nice    %sys %iowait  %idle
           1.98    0.00    0.35   11.45   86.22
avg-cpu:  %user   %nice    %sys %iowait  %idle
           1.62    0.00    0.25   34.46   63.67
```

也可以通过以下命令查询某个具体的设备块的信息，如下所示。

```
[root@localhost ~]# iostat -d -k 1 |grep sda10
Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda10              60.72      18.95       71.53  395637647 1493241908
sda10             299.02     4266.67     129.41    4352      132
sda10             483.84     4589.90    4117.17    4544     4076
sda10             218.00     3360.00     100.00    3360      100
sda10             546.00     8784.00     124.00    8784      124
sda10             827.00    13232.00     136.00   13232      136
```

2. Mpstat 工具

Mpstat 是系统实时监控工具，主要报告 CPU 的一些信息，先了解下该命令的语法：

```
mpstat [-P {|ALL|}] [internal [count]]
```

下面分析具体参数的含义：

- -P {|ALL|}: 表示需要监控哪个 CPU。
- Internal: 表示相邻的两次采样的时间间隔。
- Count: 表示采样的次数。

下面通过案例来理解一下该命令的具体用法。

(1) 显示所有 CPU 的信息，每秒钟执行一次。

```
[root@localhost ~]# mpstat -P ALL 1
Linux 2.6.25-14.fc9.i686 (localhost.localdomain)      11/23/2012      _i686_
(1 CPU)
```

| 07:49:21 AM | CPU | %usr | %nice | %sys | %iowait | %irq | %soft | %steal | %guest |
|-------------|-----|------|-------|------|---------|------|-------|--------|--------|
| idle | | | | | | | | | |
| 07:49:22 AM | all | 0.99 | 0.00 | 0.99 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 98.02 | | | | | | | | | |
| 07:49:22 AM | 0 | 0.99 | 0.00 | 0.99 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 98.02 | | | | | | | | | |
| 07:49:22 AM | CPU | %usr | %nice | %sys | %iowait | %irq | %soft | %steal | %guest |
| idle | | | | | | | | | |
| 07:49:23 AM | all | 3.03 | 0.00 | 3.03 | 0.00 | 0.00 | 1.01 | 0.00 | 0.00 |
| 92.93 | | | | | | | | | |
| 07:49:23 AM | 0 | 3.03 | 0.00 | 3.03 | 0.00 | 0.00 | 1.01 | 0.00 | 0.00 |
| 92.93 | | | | | | | | | |

(2) 显示 ID 为 0 的 CPU 的信息，每秒钟执行一次。

```
[root@localhost ~]# mpstat -P 0 1
Linux 2.6.25-14.fc9.i686 (localhost.localdomain)      11/23/2012      _i686_      (1
CPU)
```

| 10:42:38 PM | CPU | %user | %nice | %system | %iowait | %irq | %soft | %idle |
|-------------|-----|-------|-------|---------|---------|------|-------|-------|
| intr/s | | | | | | | | |
| 10:42:43 PM | all | 6.89 | 0.00 | 44.76 | 0.10 | 0.10 | 0.10 | 48.05 |
| 1121.60 | | | | | | | | |
| 10:42:43 PM | 0 | 9.20 | 0.00 | 49.00 | 0.00 | 0.00 | 0.20 | 41.60 |
| 413.00 | | | | | | | | |
| 10:42:43 PM | 1 | 4.60 | 0.00 | 40.60 | 0.00 | 0.20 | 0.20 | 54.60 |
| 708.40 | | | | | | | | |
| 10:42:43 PM | CPU | %user | %nice | %system | %iowait | %irq | %soft | %idle |
| intr/s | | | | | | | | |
| 10:42:48 PM | all | 7.60 | 0.00 | 45.30 | 0.30 | 0.00 | 0.10 | 46.70 |
| 1195.01 | | | | | | | | |
| 10:42:48 PM | 0 | 4.19 | 0.00 | 2.20 | 0.40 | 0.00 | 0.00 | 93.21 |
| 1034.53 | | | | | | | | |
| 10:42:48 PM | 1 | 10.78 | 0.00 | 88.22 | 0.40 | 0.00 | 0.00 | 0.20 |
| 160.48 | | | | | | | | |
| Average: | CPU | %user | %nice | %system | %iowait | %irq | %soft | %idle |
| intr/s | | | | | | | | |
| Average: | all | 7.25 | 0.00 | 45.03 | 0.20 | 0.05 | 0.10 | 47.38 |
| 1158.34 | | | | | | | | |

| | | | | | | | | |
|----------|---|------|------|-------|------|------|------|-------|
| Average: | 0 | 6.69 | 0.00 | 25.57 | 0.20 | 0.00 | 0.10 | 67.43 |
| 724.08 | | | | | | | | |
| Average: | 1 | 7.69 | 0.00 | 64.44 | 0.20 | 0.10 | 0.10 | 27.37 |
| 434.17 | | | | | | | | |

该结果的具体参数项的含义如下所示。

- %user: 表示处理用户进程所使用 CPU 的百分比。
- %nice: 表示使用 nice 命令对进程进行降级时 CPU 的百分比。
- %sys: 表示内核进程使用的 CPU 百分比。
- %iowait: 表示等待进行 I/O 所使用的 CPU 时间百分比。
- %irq: 表示用于处理系统中断的 CPU 百分比。
- %soft: 表示用于软件中断的 CPU 百分比。
- %idle: 显示 CPU 的空闲时间。
- %intr/s: 显示每秒 CPU 接收的中断总数。

3. sar 工具

sar 是目前 Linux 最为全面的系统性能分析工具之一，可以从多方面对系统的活动进行报告，包括：文件的读写情况、系统调用的使用情况、磁盘 I/O、CPU 效率、内存使用状况、进程活动及 IPC 有关的活动等。

sar 的语法格式如下：

```
sar [options] [-A] [-o file] t [n]
```

其中：t 为采样间隔，n 为采样次数，默认值是 1；-o file 表示将命令结果以二进制格式存放在文件中，file 是文件名。options 为命令行选项，选项的含义如下：

- -A: 所有报告的总和。
- -u: 输出 CPU 使用情况的统计信息。
- -v: 输出 inode、文件和其他内核表的统计信息。
- -d: 输出每一个块设备的活动信息。
- -r: 输出内存和交换空间的统计信息。
- -b: 显示 I/O 和传送速率的统计信息。
- -a: 文件读写情况。
- -c: 输出进程统计信息，每秒创建的进程数。
- -R: 输出内存页面的统计信息。
- -y: 终端设备活动情况。
- -w: 输出系统交换活动信息。

下面通过一个案例来理解 sar 工具的使用方法，如下所示。

```
[root@localhost ~]# sar -u 1 5
```

```
Linux 2.6.25-14.fc9.i686 (localhost.localdomain)      11/24/2012      _i686
(1 CPU)
```

| 02:33:56 AM | CPU | %user | %nice | %system | %iowait | %steal | %idle |
|-------------|-----|-------|-------|---------|---------|--------|-------|
| 02:33:57 AM | all | 2.06 | 0.00 | 5.15 | 0.00 | 0.00 | 92.78 |
| 02:33:58 AM | all | 3.00 | 0.00 | 6.00 | 0.00 | 0.00 | 91.00 |
| 02:33:59 AM | all | 41.41 | 0.00 | 9.09 | 0.00 | 0.00 | 49.49 |
| 02:34:00 AM | all | 96.77 | 0.00 | 3.23 | 0.00 | 0.00 | 0.00 |

结果中具体参数的含义如下所示。

- CPU: all 表示统计信息为所有 CPU 的平均值。
- %user: 显示在用户级别 (application) 运行使用 CPU 总时间的百分比。
- %nice: 显示在用户级别, 用于 nice 操作所占用 CPU 总时间的百分比。
- %system: 在核心级别 (kernel) 运行所使用 CPU 总时间的百分比。
- %iowait: 显示用于等待 I/O 操作占用 CPU 总时间的百分比。
- %steal: 管理程序 (hypervisor) 为另一个虚拟进程提供服务而等待虚拟 CPU 的百分比。
- %idle: 显示 CPU 空闲时间占用 CPU 总时间的百分比。

提示

如果%iowait的值过高,表示硬盘存在I/O瓶颈;如果%idle的值高但系统响应慢时,有可能是CPU等待分配内存,此时应加大内存容量;若%idle的值持续低于10,则系统的CPU处理能力相对较低,表明系统中最需要解决的资源是CPU。

9.2 开源监控利器 Nagios 实战

Nagios 是一个用来监控主机、服务和网络的开源软件。在实际工作中需要监控的对象主要是主机资源监控和网络服务监控。主机资源监控可以包括监控系统负载、当前 ip 链接数、磁盘空间使用情况、当前进程数以及自定义的资源监控等;网络服务监控可包括主机存活检查、Web 服务监控、FTP 服务监控、数据库服务监控、自定义服务器监控等。

9.2.1 安装 Nagios 之前的准备工作

在安装 Nagios 之前,需要做以下必要的工作。

步骤 01 使用 Fedora 操作系统,本例使用的是 Fedora14,并且在安装 Nagios 时拥有 root 使用权。

步骤 02 在 root 用户下面进行安装 Apache 服务器,首先下载 httpd-2.2.20.tar.gz 安装包,解压复制到/user/src/目录下。

```
[root@localhost ~]# mv httpd-2.2.0.tar.gz /usr/src
[root@localhost ~]# gunzip httpd-2.2.0.tar.gz
[root@localhost ~]# tar -xvf httpd-2.2.0.tar
[root@localhost ~]# cd /usr/src
[root@localhost src]# ls
httpd-2.2.0  httpd-2.2.0.tar  redhat
[root@localhost src]# cd httpd-2.2.0
[root@localhost httpd-2.2.0]# ls
ABOUT_APACHE      configure          LAYOUT            README
acinclude.m4        configure.in       libhttpd.dsp      README.platforms
Apache.dsw          docs              LICENSE            ROADMAP
apachenw.mcp.zip    emacs-style       Makefile.in       server
build               httpd.dsp         Makefile.win      src/lib
BuildBin.dsp        httpd.spec        modules           support
buildconf           include           NOTICE           test
CHANGES            INSTALL           NWGNUmakefile     VERSIONING
config.layout       InstallBin.dsp    os
```

```
[root@localhost httpd-2.2.0]# ./configure --prefix=/usr/local/apache
[root@localhost httpd-2.2.0]# make
[root@localhost httpd-2.2.0]# make install
```

步骤 03 执行 `/usr/local/apache/bin/apachectl -t` 命令，检测 apache 的配置文件语法是否正确，如下所示。

```
[root@localhost ~]# cd /usr/local/apache/bin/
[root@localhost bin]# ls
ab          apu-1-config  dbmmanage     htcacheclean  htpasswd  logresolve
apachectl   apxs          envvars       htdbm          httpd      rotatelog
apr-1-config  checkgid     envvars-std   htdigest      httpd2dbm
[root@localhost bin]# ./apachectl -t
Syntax OK
```

步骤 04 修改 `/etc/profile` 文件，修改 `PATH` 路径，直接把 `/usr/local/apache/bin` 的路径加入到环境变量 `PATH` 中，方便执行 apache 的命令。

```
[root@localhost ~]# vi /etc/profile
```

步骤 05 在该文件的最后直接添加下面两句设置环境变量的语句，如下所示。

```
PATH=$PATH:/usr/local/apache/bin
export PATH
```

这样可以直接运行 `apachectl start` 命令，从而启动 apache 服务了，如下所示。

```
[root@localhost ~]# apachectl start
```

接下来，可以在浏览器中输入 `http://ip`，测试 apache 服务是否成功启动。

步骤 06 将 Apache 服务设定为系统启动服务，如下所示。

```
[root@localhost ~]#chkconfig httpd on
```

9.2.2 安装 Nagios 主程序

用户可以登录 <http://www.nagios.org> 页面中下载 Nagios，然后进行安装。本案例以安装 nagios-3.4.1.tar.gz 为例进行讲解，操作步骤如下：

步骤 01 解压 nagios-3.4.1.tar.gz，如下所示。

```
[root@localhost ~]# useradd nagios
[root@localhost ~]# mkdir /usr/local/nagios
[root@localhost ~]# chown nagios.nagios /usr/local/nagios
[root@localhost ~]# ls
anaconda-ks.cfg          MySQL Cluster.txt
Desktop                  nagios
Documents                 nagios-3.4.1.tar.gz
Download                 network.txt
install.log              node.txt
install.log.syslog       Pictures
libaio-0.3.96-3.i386.rpm Public
Music                    Templates
my.cnf.bak               Unsaved Document 1
Mysql5.5                 Videos
mysql-cluster-gpl-7.2.8-linux2.6-i686.tar
[root@localhost ~]# tar zxvf nagios-3.4.1.tar.gz
```

步骤 02 开始进行编译，如下所示。

```
[root@localhost ~]# cd nagios
[root@localhost nagios]# ./configure --prefix=/usr/local/nagios
[root@localhost nagios]# make all
```

步骤 03 Nagios 主程序编译成功后，如果在编译过程中没有发生错误，接下来进行安装，如下所示。

```
[root@localhost nagios]# make install
[root@localhost nagios]# make install-config
[root@localhost nagios]# make install-init
[root@localhost nagios]# make install-commandmode
[root@localhost nagios]# ls /usr/local/nagios
bin etc libexec sbin share var
```

Nagios 服务器安装完成后，在安装目录/usr/local/nagios 下生成下面的目录，如表 9-1 所示。

表 9-1 Nagios 服务器目录的具体作用

| | |
|-------|------------------------------------------|
| bin | Nagios 执行程序所在目录，这个目录只有一个文件 nagios |
| etc | Nagios 配置文件位置，初始安装完后，只有几个*.cfg-sample 文件 |
| sbin | Nagios Cgi 文件所在目录，也就是执行外部命令所需文件所在的目录 |
| share | Nagios 网页文件所在的目录 |
| var | Nagios 日志文件、spid 等文件所在的目录 |

步骤 04 将 nagios 的启动脚本添加到系统服务当中去，如下所示。

```
[root@localhost ~]# vi /etc/profile
```

步骤 05 在该文件的最后直接添加下面两句设置环境变量的语句，如下所示。

```
PATH=$PATH:/usr/local/nagios/bin
export PATH
[root@localhost ~]# chkconfig --add nagios
```

步骤 06 下面执行以下命令，将 nagios 添加到系统服务中去。

```
[root@localhost ~]# chkconfig nagios on
```

9.2.3 整合 Nagios 到 Apache 服务

Nagios 安装完成后，即可将其添加到 Apache 服务中，具体操作步骤如下：

步骤 01 将 Apache 服务宿主用户加入到 Nagios 组里面，这样做的目的是为了 Apache 有适当的权限通过 CGI 命令对 Nagios 进行调用，例如通过浏览器界面关闭 Nagios 服务。

```
[root@localhost ~]# cd /usr/local/nagios/
[root@localhost nagios]# usermode -G nagios apache
[root@localhost nagios]# usermod -G nagios apache
```

步骤 02 打开/etc/httpd/conf/httpd.conf 文件，Apache 的运行用户和运行组默认的是 daemon，把 Apache 的运行用户和运行组改成 nagios。

```
User nagios
Group nagios
```

步骤 03 将 Nagios 的信息加入到 Apache 中，打开/etc/httpd/conf/httpd.conf 文件，在文件最后添加如下代码：

```
#setting for nagios
ScriptAlias /nagios/cgi-bin /usr/local/nagios/sbin
<Directory "/usr/local/nagios/sbin">
AuthType Basic
Options ExecCGI
AllowOverride None
Order allow,deny
```

```

Allow from all
AuthName "Nagios Access"
AuthUserFile /usr/local/nagios/etc/htpasswd
Require valid-user
</Directory>

Alias /nagios /usr/local/nagios/share
<Directory "/usr/local/nagios/share">
AuthType Basic
Options None
AllowOverride None
Order allow,deny
Allow from all
    AuthName "nagios Access"
AuthUserFile /usr/local/nagios/etc/htpasswd
Require valid-user
</Directory>

```

上述文本的作用主要是用来对 Nagios 目录进行用户验证，只有合法的用户才能访问 Nagios 的页面，配置完成之后，可以执行 `apachectl -t` 检测 apache 配置文件的语法是否有错误。

```

[root@localhost ~]# apachectl -t
Syntax OK

```

步骤 04 创建 Web 接口，如下所示。

```

[root@localhost ~]# cd nagios
[root@localhost nagios]# make install-webconf
/usr/bin/install -c -m 644 sample-config/httpd.conf
/etc/httpd/conf.d/nagios.conf

```

```

*** Nagios/Apache conf file installed ***

```

步骤 05 创建一个用于登录 Nagios 的用户名和密码，如下所示。

```

[root@localhost ~]# /usr/bin/htpasswd -c /usr/local/nagios/etc/htpasswd.users
nagios
New password:
Re-type new password:
Adding password for user nagios

```

步骤 06 重启 Apache 服务，重启 Nagios 服务，如下所示：

```

[root@localhost ~]# httpd -k restart
[root@localhost ~]# service nagios restart
Running configuration check...done.
Stopping nagios: done.
Starting nagios: done.

```


步骤 07 从另外一台计算机上登录 `http://ip/nagios`，此时会要求输入上面创建的用户名和密码，如图 9-1 所示。

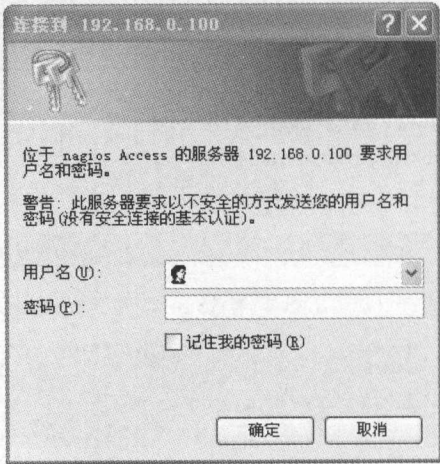


图 9-1 登录 nagios 服务器验证窗口

步骤 08 输入正确的用户名和密码之后，单击【确定】按钮，即可看到 Nagios 服务器主页面，如图 9-2 所示。

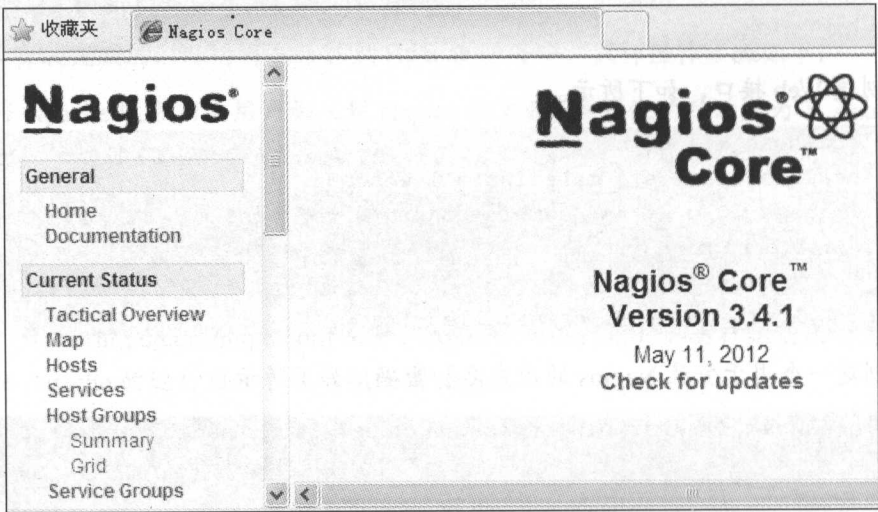


图 9-2 nagios 服务器主页面

配置 nagios 的时候，经常会遇到提示 Internal Server Error 错误。错误如下：

Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request.

Please contact the server administrator, `root@localhost` and inform them of the time the error occurred, and anything you might have done that may have caused the error.

More information about this error may be available in the server error log.

此时查看下 Apache 服务器的错误日志信息，如下所示。

```
[root@localhost ~]# vi /var/log/httpd/error_log
[Tue Nov 20 09:33:30 2012] [error] [client 192.168.0.208] (13)Permission denied:
exec of '/usr/local/nagios/sbin/status.cgi' failed, referer:
http://192.168.0.100/nagios/side.php
[Tue Nov 20 09:33:30 2012] [error] [client 192.168.0.208] Premature end of script
headers: status.cgi, referer: http://192.168.0.100/nagios/side.php
[Tue Nov 20 09:34:49 2012] [error] [client 192.168.0.208] PHP Warning:
MagpieRSS: Failed to fetch http://www.nagios.org/backend/feeds/frontpage/ and
cache is off in /usr/local/nagios/share/includes/rss/rss_fetch.inc on line 238,
referer: http://192.168.0.100/nagios/main.php
```

错误的含义是 Apache 开启了 suexec 的功能，对执行 CGI 的路径进行了限制。先检测一下计算机是的 Apache 是否打开了 suexec 功能，如下所示。

```
[root@localhost sysconfig]# httpd -V
Server version: Apache/2.2.8 (Unix)
Server built: Feb 25 2008 07:05:32
Server's Module Magic Number: 20051115:11
Server loaded: APR 1.2.12, APR-Util 1.2.12
Compiled using: APR 1.2.12, APR-Util 1.2.12
Architecture: 32-bit
Server MPM: Prefork
    threaded: no
    forked: yes (variable process count)
Server compiled with....
-D APACHE_MPM_DIR="server/mpm/prefork"
-D APR_HAS_SENDFILE
-D APR_HAS_MMAP
-D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
-D APR_USE_SYSVSEM_SERIALIZE
-D APR_USE_PTHREAD_SERIALIZE
-D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
-D APR_HAS_OTHER_CHILD
-D AP_HAVE_RELIABLE_PIPED_LOGS
-D DYNAMIC_MODULE_LIMIT=128
-D HTTPD_ROOT="/etc/httpd"
-D SUEXEC_BIN="/usr/sbin/suexec"
-D DEFAULT_PIDLOG="logs/httpd.pid"
-D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
-D DEFAULT_LOCKFILE="logs/accept.lock"
-D DEFAULT_ERRORLOG="logs/error_log"
-D AP_TYPES_CONFIG_FILE="conf/mime.types"
-D SERVER_CONFIG_FILE="conf/httpd.conf"
```

从结果可以看出, Apache 已经启用了 suexec 功能。执行下列命令, 可以知道 Apache 如何限制 CGI 执行的路径, 如下所示。

```
[root@localhost sysconfig]# suexec -V
-D AP_DOC_ROOT="/var/www"
-D AP_GID_MIN=100
-D AP_HTTPD_USER="apache"
-D AP_LOG_EXEC="/var/log/httpd/suexec.log"
-D AP_SAFE_PATH="/usr/local/bin:/usr/bin:/bin"
-D AP_UID_MIN=500
-D AP_USERDIR_SUFFIX="public_html"
```

CGI 程序只能在 /var/www/ 目录下面才能够正常运行, 因为将 Nagios 安装在 /usr/local/nagios 目录下面的, 所以 CGI 程序无法执行。最简单的解决方法就是将 Nagios 安装到 /var/www/nagios 就可以了。值得注意的是, 编译的时候需要指定—prefix=/var/www/nagios, 其他编译安装选项不用改变。

如果不想重新编译, 可以将 /usr/local 目录下面的 nagios 移动到 /var/www 目录下面, 然后修改相关的配置文件。值得注意的是, nagios 目录下面所有的 cfg 的路径都需要修改, 所以建议读者还是重新编译安装, etc/profile 环境变量也要重新设置。笔者在这里重新安装一遍 Nagios, 上述问题解决。

9.2.4 安装 Nagios 插件包

安装 Nagios 插件包的具体操作步骤如下:

步骤 01 下载 nagios 插件包, 然后解压缩, 如下所示。

```
[root@localhost ~]# tar -zxvf nagios-plugins-1.4.16.tar.gz
[root@localhost ~]# cd nagios-plugins-1.4.16
```

步骤 02 编译并且开始安装 nagios 插件, 如下所示。

```
[root@localhost nagios-plugins-1.4.16]# ./configure --with-nagios-user=nagios
--with-nagios-group=nagios -prefix=/var/www/nagios
[root@localhost nagios-plugins-1.4.16]# make
[root@localhost nagios-plugins-1.4.16]# make install
```

步骤 03 验证 nagios 的样例配置文件, 如下所示。

```
[root@localhost nagios-plugins-1.4.16]# /var/www/nagios/bin/nagios -v
/var/www/nagios/etc/nagios.cfg
```

```
Nagios Core 3.4.1
Copyright (c) 2009-2011 Nagios Core Development Team and Community Contributors
Copyright (c) 1999-2009 Ethan Galstad
Last Modified: 05-11-2012
```



```
License: GPL

Website: http://www.nagios.org
Reading configuration data...
  Read main config file okay...
Processing object config file '/var/www/nagios/etc/objects/commands.cfg'...
Processing object config file '/var/www/nagios/etc/objects/contacts.cfg'...
Processing object config file
'/var/www/nagios/etc/objects/timeperiods.cfg'...
Processing object config file '/var/www/nagios/etc/objects/templates.cfg'...
Processing object config file '/var/www/nagios/etc/objects/localhost.cfg'...
  Read object config files okay...

Running pre-flight check on configuration data...

Checking services...
  Checked 8 services.
Checking hosts...
  Checked 1 hosts.
Checking host groups...
  Checked 1 host groups.
Checking service groups...
  Checked 0 service groups.
Checking contacts...
  Checked 1 contacts.
Checking contact groups...
  Checked 1 contact groups.
Checking service escalations...
  Checked 0 service escalations.
Checking service dependencies...
  Checked 0 service dependencies.
Checking host escalations...
  Checked 0 host escalations.
Checking host dependencies...
  Checked 0 host dependencies.
Checking commands...
  Checked 24 commands.
Checking time periods...
  Checked 5 time periods.
Checking for circular paths between hosts...
Checking for circular host and service dependencies...
Checking global event handlers...
Checking obsessive compulsive processor commands...
Checking misc settings...
```

```
Total Warnings: 0
```

```
Total Errors: 0
```

```
Things look okay - No serious problems were detected during the pre-flight check
[root@localhost nagios-plugins-1.4.16]#
```

步骤 04 重新启动 nagios 服务和 apache 服务，如下所示。

```
[root@localhost nagios-plugins-1.4.16]# service nagios restart
Running configuration check...done.
Stopping nagios: done.
Starting nagios: done.
[root@localhost nagios-plugins-1.4.16]# service httpd restart
Stopping httpd: [ OK ]
Starting httpd: [ OK ]
```

9.2.5 监控服务器的 CPU、负载、磁盘 I/O 使用情况

在监控服务器之前，需要在服务器中安装 nrpe，配置 nagios 以及使用 htpasswd 创建浏览器验证账号。具体操作步骤如下：

步骤 01 在 MySQL 服务器上安装 nrpe，如下所示。

```
[root@localhost ~]# tar -zxvf nrpe-2.12.tar.gz
[root@localhost ~]# tar -zxvf nrpe-2.12.tar.gz
[root@localhost nrpe-2.12]# ./configure --prefix=/var/www/nagios
[root@localhost nrpe-2.12]# make all
[root@localhost nrpe-2.12]# make install-plugin
[root@localhost nrpe-2.12]# make install-daemon
[root@localhost nrpe-2.12]# make install-daemon-config
[root@localhost nrpe-2.12]# make install-xinetd
```

步骤 02 查看 nrpe 的一些系统信息，比如端口 5666，可以通过以下操作修改端口。

```
[root@localhost ~]# vi /etc/xinetd.d/nrpe
# description: NRPE (Nagios Remote Plugin Executor)
service nrpe
{
    flags            = REUSE
    socket_type      = stream
    port             = 5666
    wait             = no
    user             = nagios
    group            = nagios
    server            = /var/www/nagios/bin/nrpe
    server_args       = -c /var/www/nagios/etc/nrpe.cfg --inetd
    log_on_failure   += USERID
```

```

disable      = no
only_from    = 127.0.0.1
}

```

步骤 03 在/var/www/nagios/etc/objects/command.cfg 命令定义文件中添加 nrpe 命令，如下所示：

```
[root@localhost ~]# vi /var/www/nagios/etc/objects/command.cfg
```

添加 NRPE 功能命令。

```

define command(
    command_name    check_nrpe
    command_line     $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
)

```

这里有几点需要说明：

(1) 该命令的名字叫作 check_nrpe。

(2) \$USER1\$/check_nrpe 会通过应用 resource.cfg 获得 \$USER1\$ 变量的路径。从而可以获得 /var/www/nagios/libexec/check_nrpe 这个绝对路径。

(3) -H \$HOSTADDRESS\$ 用来获得指定被检测主机的 IP 地址。

(4) -c \$ARG1\$ 用来指定被检测主机上 NRPE 守护进程运行着的 NRPE 命令名。

步骤 04 修改客户端配置，如下所示。

```

vi /etc/nagios/nrpe.cfg
#下列配置表示允许127.0.0.1, 192.168.121.55 (Server) 这两台机器访问当前机器的信息。
allowed_hosts=127.0.0.1,192.168.121.55

```

步骤 05 测试客户端配置情况，如果没有启动 nrpe 服务，会提示 “Connection refused by host” 信息。

```

#启动 nrpe (Nagios Remote Plugin Executor)
[root@localhost ~]# cd /var/www/nagios/bin
[root@localhost bin]# ./nrpe -c /var/www/nagios/etc/nrpe.cfg -d
[root@localhost bin]# netstat -an|grep 5666
tcp        0      0 0.0.0.0:5666          0.0.0.0:*            LISTEN
unix 3 [ ]  STREAM  CONNECTED  15666  /tmp/orbit-root/linc-aad-0-6c80032044a1a

```

步骤 06 接着测试下是否可以监控客户端，如下所示。

```

[root@localhost ~]# cd /var/www/nagios/libexec
[root@localhost libexec]# ./check_nrpe -H localhost -c check_load
OK - load average: 0.06, 0.03, 0.12|load1=0.060;15.000;30.000;0;
load5=0.030;10.000;25.000;0; load15=0.120;5.000;20.000;0;

```

步骤 07 在 Nagios 监控服务器上配置服务端的信息，如下所示。

```
# 创建 client 配置 (192.168.0.100为 client ip)
```



```
[root@localhost ~]# vi /var/www/nagios/etc/objects/192.168.0.100.cfg
define host{
    use linux-server
    host_name 127.0.0.1
alias client-1
    address 127.0.0.1
}

define service{
    use generic-service
host_name 127.0.0.1
    service_description check_disk2
    check_command check_nrpe!check_sda2
}

define service{
    use generic-service
host_name 127.0.0.1
    service_description check_load
    check_command check_nrpe!check_load
}

define service{
    use generic-service
    host_name 127.0.0.1
    service_description check_swap
    check_command check_nrpe!check_swap
}
```

步骤 08 将上面配置加到系统中，如下所示。

```
[root@localhost ~]# vi /var/www/nagios/etc/nagios.cfg
cfg_file=/etc/nagios/objects/192.168.0.100.cfg
```

步骤 09 重新启动 nagios 服务和 apache 服务，如下所示。

```
[root@localhost nagios-plugins-1.4.16]# service nagios restart
Running configuration check...done.
Stopping nagios: done.
Starting nagios: done.
[root@localhost nagios-plugins-1.4.16]# service httpd restart
Stopping httpd: [ OK ]
Starting httpd: [ OK ]
```

提示

如果 nagios web 界面提示如下错误信息:

It appears as though you do not have permission to view information for any of the services you requested...

可以打开 `cgi.cfg` 配置文件, 里面有个参数:

```
use_authentication=1
```

为了保障系统的安全性, nagios 设置了这个参数, 默认为 1, 改为 0 即可解决上面的问题。

打开 Nagios 页面左侧的 **Services**, 页面右侧中显示检测的所有服务项, 如图 9-3 所示。

| Host | Service | Status | Last Check | Duration | Attempt |
|---------------|-----------------------------|----------|---------------------|----------------|---------|
| 192.168.0.100 | check_disk2 | CRITICAL | 11-21-2012 16:50:54 | 0d 0h 2m 15s | 2/3 |
| | check_load | CRITICAL | 11-21-2012 16:50:03 | 0d 0h 1m 6s | 1/3 |
| | check_swap | PENDING | N/A | 0d 0h 3m 24s+ | 1/3 |
| localhost | Currdefine service(ent Load | OK | 11-21-2012 16:49:25 | 0d 3h 41m 44s | 1/4 |
| | Current Users | OK | 11-21-2012 16:46:10 | 0d 16h 17m 27s | 1/4 |
| | HTTP | CRITICAL | 11-21-2012 16:46:25 | 0d 15h 16m 2s | 4/4 |
| | PING | OK | 11-21-2012 16:47:23 | 0d 16h 14m 20s | 1/4 |
| | Root Partition | CRITICAL | 11-21-2012 16:47:20 | 1d 5h 55m 59s | 4/4 |
| | SSH | CRITICAL | 11-21-2012 16:46:35 | 0d 15h 17m 57s | 4/4 |
| | Swap Usage | OK | 11-21-2012 16:50:29 | 0d 16h 16m 57s | 1/4 |
| | Total Processes | OK | 11-21-2012 16:50:13 | 0d 16h 15m 56s | 1/4 |
| | | | | | |

图 9-3 检测到的所有的服务项

如果提示以下错误信息: `CHECK_NRPE: Error - Could not complete SSL handshake`, 可以使用下面的步骤解决。

步骤 01 检测 `nrpe` 机器上防火墙有没有启动, 启动的话要把 5666 端口打开, 也可以直接关闭防火墙, 如下所示。

```
[root@localhost ~]# service iptables stop
```

步骤 02 看看 `nrpe` 编译的是否支持 `ssl`(`nrpe` 编译默认是支持 `ssl`), 另外 `openssl` 和 `openssl-devel` 也都要装上, 如下所示。

```
[root@localhost ~]# tar -zxvf openssl-0.9.8l.tar.gz
[root@localhost openssl-0.9.8l]# ./config --prefix=/usr/local/ssl-0.9.8l
shared zlib-dynamic enable-camellia
[root@localhost openssl-0.9.8l]# make depend
[root@localhost openssl-0.9.8l]# make
[root@localhost openssl-0.9.8l]# make test
```

```
[root@localhost openssl-0.9.81]# make install
```

步骤 03 建立连接, 如下所示。

```
[root@localhost openssl-0.9.81]# cd /usr/local/
[root@localhost local]# ln -s ssl-0.9.81 ssl
[root@localhost local]# vi /etc/ld.so.conf
include ld.so.conf.d/*.conf
/usr/local/ssl/lib

[root@localhost local]# ldconfig
```

步骤 04 配置环境变量, 在/etc/profile 文件的末尾添加以下信息。

```
[root@localhost local]# vi /etc/profile
PATH=$PATH:/var/local/ssl/bin
export PATH
```

步骤 05 测试 SSL 是否安装成功, 如下所示。

```
[root@localhost ~]# cd /usr/local/
[root@localhost local]# ls
apache doc games lib man mysql-cluster share ssl
bin etc include libexec mysql sbin src ssl-0.9.81
[root@localhost local]# ldd /usr/local/ssl/bin/openssl
linux-gate.so.1 => (0x00110000)
libssl.so.0.9.8 => /usr/local/ssl-0.9.81/lib/libssl.so.0.9.8
(0x00111000)
libcrypto.so.0.9.8 => /usr/local/ssl-0.9.81/lib/libcrypto.so.0.9.8
(0x00157000)
libdl.so.2 => /lib/libdl.so.2 (0x0090c000)
libc.so.6 => /lib/libc.so.6 (0x00776000)
/lib/ld-linux.so.2 (0x00756000)
[root@localhost local]# which openssl
/usr/bin/openssl
```

步骤 06 错误有可能是配置文件的问题, 需要修改 nrpe.cfg 文件, 把 allowed_hosts=127.0.0.1 修改为 allowed_hosts=127.0.0.1,192.168.0.100, 然后重启其 nrpe 服务。

```
[root@localhost lib]# ps -ef|grep nrpe
nagios 6591 1 0 Nov20 ? 00:00:03 ./nrpe -c
/var/www/nagios/etc/nrpe.cfg -d
root 13152 1932 0 Nov21 pts/2 00:00:00 find / nrpe
root 20793 20118 0 11:14 pts/3 00:00:00 grep nrpe
[root@localhost lib]# kill -9 6591
[root@localhost lib]# cd
[root@localhost var]# cd /var/www/nagios/bin/
[root@localhost bin]# ./nrpe -c /var/www/nagios/etc/nrpe.cfg -d
```


步骤 07 把之前在/var/www/nagios/etc/nrpe.cfg 文件中添加的两条 command 记录删除, 问题即可解决。

如果还是错误, 那么按照以下解决方法。

- (1) 确认 check_nrpe 和 nrpe daemon 的版本一定要一致。
- (2) 确认 check_nrpe 和 nrpe daemon 端同时启用或者禁用 ssl 支持。
- (3) 确认 nrep.cfg 可以被 nrpe 用户正常读取。

9.2.6 配置 Nagios 监控 MySQL 服务器

下面讲述通过配置 Nagios, 从而实现监控 MySQL 服务器的方法。具体操作步骤如下:

步骤 01 首先建立 nagios 数据库, 创建 nagios 数据库用户, 授予 nagios 用户权限, 如下所示。

```
[root@localhost ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.5.27-ndb-7.2.8-cluster-gpl-log MySQL Cluster Community
Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database nagios;
Query OK, 1 row affected (0.29 sec)

mysql> grant select on nagios.* to nagios@'%' identified by 'password';
Query OK, 0 rows affected (0.31 sec)
```

步骤 02 为了方便本地测试 nagios 服务, 授权给 nagios@'localhost', 如下所示。

```
mysql> grant select on nagios.* to nagios@'localhost' identified by 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> select User,Password,host from mysql.user;
+-----+-----+-----+
| User | Password | Host |
+-----+-----+-----+
```

```
--+
| root | | localhost |
| root | | localhost.localdomain |
| root | | 127.0.0.1 |
| root | | ::1 |
| | | localhost |
| | | localhost.localdomain |
| nagios | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 | % |
| nagios | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 | localhost |
+-----+-----+-----+
--+
8 rows in set (0.00 sec)
```

步骤 03 接下来执行 `check_mysql` 命令, 检测 nagios 是否能够连接 MySQL 服务, 如下所示。

```
[root@localhost ~]# cd /var/www/nagios/libexec/
[root@localhost libexec]# ./check_mysql -u nagios -d nagios -p password
./check_mysql: error while loading shared libraries: libmysqlclient.so.18:
cannot open shared object file: No such file or directory
```

步骤 04 此时, 发现加载 `libmysqlclient.so.18` 发生错误, 需要建立一个链接, 如下所示。

```
[root@localhost usr]# cd /usr/local/mysql/lib/
[root@localhost lib]# ls
libmysqlclient.a          libmysqlclient.so.18      libndbclient.so.6.0.0
libmysqlclient_r.a        libmysqlclient.so.18.0.0 libndbclient_static.a
libmysqlclient_r.so       libmysqld.a               libtcmalloc_minimal.so
libmysqlclient_r.so.18    libmysqld-debug.a        ndb_engine.so
libmysqlclient_r.so.18.0.0 libmysqlservices.a        plugin
libmysqlclient.so         libndbclient.so
[root@localhost lib]# ln -s /usr/local/mysql/lib/libmysqlclient.so.18
/usr/lib/libmysqlclient.so.18
```

步骤 05 接着执行 `check_mysql` 命令检测 nagios 是否能够连接 MySQL 服务, 如下所示。

```
[root@localhost ~]# cd /var/www/nagios/libexec/
[root@localhost libexec]# ./check_mysql -H 192.168.0.100 -u nagios -d nagios
-p password
Uptime: 73815 Threads: 2 Questions: 16 Slow queries: 0 Opens: 33 Flush
tables: 1 Open tables: 26 Queries per second avg: 0.000
```

步骤 06 下面开始 Nagios 配置文件, 如下所示。

```
[root@localhost ~]# cd /var/www/nagios/objects/
[root@localhost objects]# vi commands.cfg
```

步骤 07 在 `commands.cfg` 文件的末尾接上如下的文本。

```
#check_mysql
```

```
define command{
    command_name    check_mysql
    command_line    $USER1$/check_mysql -H $HOSTADDRESS$ -u nagios -d nagios -p
password
}
```

步骤 08 在 192.168.0.100.cfg 文件末尾追加 check_mysql 服务，如下所示。

```
[root@localhost ~]# vi /var/www/nagios/etc/objects/192.168.0.100.cfg
define service{
    use generic-service
    host_name 192.168.0.100
    service_description MySQL
    normal_check_interval 1
    retry_check_interval 1
    check_command check_mysql
}
```

步骤 09 打开 Nagios 左侧的页面的 services，然后选择 MySQL，可以看到信息如图 9-4 所示。

Service State Information

| | |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Current Status: | OK (for 0d 2h 7m 29s) |
| Status Information: | Uptime: 166736 Threads: 1 Questions: 135 Slow queries: 0 Opens: 33 Flush tables: 1 Open tables: 26 Queries per second avg: 0.000 |
| Performance Data: | |
| Current Attempt: | 1/3 (HARD state) |
| Last Check Time: | 11-22-2012 06:26:01 |
| Check Type: | ACTIVE |
| Check Latency / Duration: | 0.281 / 20.036 seconds |
| Next Scheduled Check: | 11-22-2012 06:27:01 |
| Last State Change: | 11-22-2012 04:19:01 |
| Last Notification: | N/A (notification 0) |
| Is This Service Flapping? | NO (0.00% state change) |
| In Scheduled Downtime? | NO |
| Last Update: | 11-22-2012 06:26:22 (0d 0h 0m 8s ago) |
| Active Checks: | ENABLED |
| Passive Checks: | ENABLED |
| Obsessing: | ENABLED |
| Notifications: | ENABLED |
| Event Handler: | ENABLED |
| Flap Detection: | ENABLED |

图 9-4 检测 MySQL 的状态信息

9.3 MySQL 监控利器 Cacti 实战

本节讲述 Cacti 工具的使用方法。

9.3.1 Cacti 工具的安装

Cacti 是一套基于 PHP、MySQL、SNMP，以及 RRDTool 开发的网络流量监测分析工具，通过 snmpget 获得数据，使用 RRDTool 绘画图形。

在安装 Cacti 工具之前，需要安装 MySQL 服务器和 Apache 服务，然后安装 PHP。安装 PHP 之前，还需要安装 zlib、freetype、libpng、jpegsrc 和 Fontconfig，从而使 PHP 支持 GD 库。GD 库的下载地址是 <http://oss.oetiker.ch/rrdtool/pub/libs/>。

(1) 安装 zlib，如下所示。

```
[root@localhost ~]# tar zxvf zlib-1.2.3.tar.gz
[root@localhost ~]# cd zlib-1.2.3
[root@localhost zlib-1.2.3]# ./configure --prefix=/usr/local/zlib
[root@localhost zlib-1.2.3]# make
[root@localhost zlib-1.2.3]# make install
```

(2) 安装 libpng，如下所示。

```
[root@localhost ~]# tar zxvf libpng-1.2.18.tar.gz
[root@localhost scripts]# mv makefile.linux ../makefile
[root@localhost scripts]# cd ..
[root@localhost libpng-1.2.18]# make
[root@localhost libpng-1.2.18]# make install
```

(3) 安装 freetype，如下所示。

```
[root@localhost ~]# tar zxvf freetype-2.3.5.tar.gz
[root@localhost ~]# cd freetype-2.3.5
[root@localhost freetype-2.3.5]# ./configure --prefix=/usr/local/freetype
[root@localhost freetype-2.3.5]# make
[root@localhost freetype-2.3.5]# make install
```

(4) 安装 jpegsrc，如下所示。

```
[root@localhost ~]# tar -zxvf jpegsrc.v6b.tar.gz
[root@localhost ~]# cd jpeg-6b/
[root@localhost jpeg-6b]# mkdir /usr/local/libjpeg
[root@localhost jpeg-6b]# mkdir /usr/local/libjpeg/include
[root@localhost jpeg-6b]# mkdir /usr/local/libjpeg/bin
[root@localhost jpeg-6b]# mkdir /usr/local/libjpeg/lib
[root@localhost jpeg-6b]# mkdir /usr/local/libjpeg/man
[root@localhost jpeg-6b]# mkdir /usr/local/libjpeg/man/man1
[root@localhost jpeg-6b]# ./configure --prefix=/usr/local/libjpeg
--enable-shared --enable-static
[root@localhost jpeg-6b]# make
[root@localhost jpeg-6b]# make install
```

(5) 安装 Fontconfig，如下所示。

```
[root@localhost ~]# tar -zxvf fontconfig-2.4.2.tar.gz
[root@localhost fontconfig-2.4.2]# ./configure
--with-freetype-config=/usr/local/freetype
[root@localhost fontconfig-2.4.2]# make
[root@localhost fontconfig-2.4.2]# make install
```

(6) 安装 GD, 如下所示。

```
[root@localhost ~]# tar -zxvf gd-2.0.35.tar.gz
[root@localhost ~]# cd gd-2.0.35
[root@localhost gd-2.0.35]# ./configure --prefix=/usr/local/libgd --with-png
--with-freetype=/usr/local/freetype --with-jpeg=/usr/local/libjpeg
[root@localhost gd-2.0.35]# make
[root@localhost gd-2.0.35]# make install
```

(7) 安装 PHP, 如下所示。

```
[root@localhost ~]# tar -zxvf php-5.2.3.tar.gz
[root@localhost ~]# cd php-5.2.3
[root@localhost php-5.2.3]# ./configure --prefix=/usr/local/php
--with-apxs2=/usr/local/apache/bin/apxs --with-mysql=/usr/local/mysql
--with-gd=/usr/local/libgd --enable-gd-native-ttf --with-ttf
--enable-gd-jis-conv --with-freetype-dir=/usr/local/freetype
--with-jpeg-dir=/usr/local/libjpeg --with-png-dir=/usr
--with-zlib-dir=/usr/local/zlib --enable-xml --enable-mbstring --enable-sockets
[root@localhost php-5.2.3]# make
[root@localhost php-5.2.3]# make install
[root@localhost php-5.2.3]# cp php.ini-recommended /usr/local/php/lib/php.ini
[root@localhost php-5.2.3]# ln -s /usr/local/php/bin/* /usr/local/bin/
```

(8) 下面需要修改 httpd.conf 文件, 如下所示。

```
[root@localhost ~]# vi /usr/local/apache/conf/httpd.conf
```

在这个文件中查找如下语句:

```
AddType application/x-compress .Z
AddType application/x-gzip .gz .tgz
```

在其下加入:

```
AddType application/x-tar .tgz
AddType application/x-httpd-php .php
AddType image/x-icon .ico
```

修改 DirectoryIndex 行, 添加 index.php, 如下:

```
DirectoryIndex index.php index.html index.html.var
```

(9) 最后创建 test.php 进行测试是否成功安装 php, 如下所示。

```
[root@localhost ~]# vi /usr/local/apache/htdocs/test.php
```

添加以下行，添加完成以后，保存退出。


```
<?php
phpinfo();
?>
```

重启 apache 服务器，如下所示。

```
[root@localhost ~]# /usr/local/apache/bin/httpd -k stop
[root@localhost ~]# /usr/local/apache/bin/httpd -k start
```

在浏览器中输入：http://192.168.0.100/test.php 进行 PHP 测试，如图 9-5 所示。

PHP Version 5.2.3



| | |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| System | Linux localhost.localdomain 2.6.25-14.fc9.i686 #1 SMP Thu May 1 06:28:41 EDT 2008 i686 |
| Build Date | Nov 26 2012 01:38:31 |
| Configure Command | ./configure '--prefix=/usr/local/php' '--with-apxs2=/usr/local/apache/bin/apxs' '--with-mysql=/usr/local/mysql' '--with-gd=/usr/local/libgd' '--enable-gd-native-ttf' '--with-ttf' '--enable-gd-jis-conv' '--with-freetype-dir=/usr/local/freetype' '--with-jpeg-dir=/usr/local/libjpeg' '--with-png-dir=/usr' '--with-zlib-dir=/usr/local/zlib' '--enable-xml' '--enable-mbstring' '--enable-sockets' |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | disabled |
| Configuration File (php.ini) Path | /usr/local/php/lib |
| Loaded Configuration File | /usr/local/php/lib/php.ini |
| PHP API | 20041225 |
| PHP Extension | 20060613 |
| Zend Extension | 220060519 |
| Debug Build | no |
| Thread Safety | disabled |
| Zend Memory Manager | enabled |
| IPv6 Support | enabled |

图 9-5 PHP 测试信息页面

(10) 下面开始安装 RRDTool。安装 rrdtool-1.2.11 需要一些库文件支持，需要将 cgilib-0.5.tar.gz, zlib-1.2.2.tar.gz, libpng-1.2.8.tar.gz, freetype-2.1.9.tar.gz, libart_lgpl-2.3.17.tar.gz, rrdtool-1.2.11.tar.gz 放到 /root/rrdtool-1.2.11 目录下，将脚本保存为 /root/rrdtool-1.2.11/rrdtoolinstall.sh，并给执行权限 chmod u+x /root/rrdtool-1.2.11/rrdtoolinstall.sh, rrdtoolinstall.sh 脚本信息如下所示。

```
#!/bin/sh
BUILD_DIR=/root/rrdtool-1.2.11
INSTALL_DIR=/usr/local/rrdtool
cd $BUILD_DIR
```



```

tar xzf cgilib-0.5.tar.gz
cd cgilib-0.5
make CC=gcc CFLAGS="-O3 -fPIC -I."
mkdir -p $BUILD_DIR/lb/include
cp *.h $BUILD_DIR/lb/include
mkdir -p $BUILD_DIR/lb/lib
cp libcgi* $BUILD_DIR/lb/lib
cd $BUILD_DIR
tar xzf zlib-1.2.2.tar.gz
cd zlib-1.2.2
env CFLAGS="-O3 -fPIC" ./configure --prefix=$BUILD_DIR/lb
make
make install
cd $BUILD_DIR
tar zxvf libpng-1.2.8-config.tar.gz
cd libpng-1.2.8-config
env CPPFLAGS="-I$BUILD_DIR/lb/include" LDFLAGS="-L$BUILD_DIR/lb/lib"
CFLAGS="-O3 -fPIC" \
    ./configure --disable-shared --prefix=$BUILD_DIR/lb
make
make install
cd $BUILD_DIR
tar zxvf freetype-2.1.9.tar.gz
cd freetype-2.1.9
env CPPFLAGS="-I$BUILD_DIR/lb/include" LDFLAGS="-L$BUILD_DIR/lb/lib"
CFLAGS="-O3 -fPIC" \
    ./configure --disable-shared --prefix=$BUILD_DIR/lb
make
make install

cd $BUILD_DIR
tar zxvf libart_lgpl-2.3.17.tar.gz
cd libart_lgpl-2.3.17
env CFLAGS="-O3 -fPIC" ./configure --disable-shared --prefix=$BUILD_DIR/lb
make
make install

IR=-I$BUILD_DIR/lb/include
CPPFLAGS="$IR $IR/libart-2.0 $IR/freetype2 $IR/libpng"
LDFLAGS="-L$BUILD_DIR/lb/lib"
CFLAGS=-O3
export CPPFLAGS LDFLAGS CFLAGS

cd $BUILD_DIR
tar xzf rrdtool-1.2.11.tar.gz
cd rrdtool-1.2.11
./configure --prefix=$INSTALL_DIR --disable-python --disable-tcl && make &&
make install

```

(11) 直接运行 rrdtoolinstall.sh 脚本即可，如下所示。

```

[root@localhost ~]# tar -zxvf rrdtool-1.2.11.tar.gz
[root@localhost ~]# cd rrdtool-1.2.11
[root@localhost rrdtool-1.2.11]# ls

```

```
cgilib-0.5.tar.gz      libpng-1.2.8-config.tar.gz  zlib-1.2.2.tar.gz
freetype-2.1.9.tar.gz  rrdtool-1.2.11.tar.gz
libart_lgpl-2.3.17.tar.gz  rrdtoolinstall.sh
[root@localhost rrdtool-1.2.11]# chmod u+x rrdtoolinstall.sh
[root@localhost rrdtool-1.2.11]# ./rrdtoolinstall.sh
```

(12) 安装 net-snmp，如下所示。

```
[root@localhost ~]# tar -zxvf net-snmp-5.2.4.tar.gz
[root@localhost ~]# cd net-snmp-5.2.4
[root@localhost net-snmp-5.2.4]# ./configure --prefix=/usr/local/net-snmp
--enable-developer
[root@localhost ~]# make
[root@localhost ~]# make install
```

(13) 下面修改 snmpd.conf，然后启动 snmpd 服务。

```
[root@localhost net-snmp-5.2.4]# ln -s /usr/local/net-snmp/bin/*
/usr/local/bin/
[root@localhost net-snmp-5.2.4]# cp EXAMPLE.conf
/usr/local/net-snmp/share/snmp/snmpd.conf
[root@localhost net-snmp-5.2.4]# /usr/local/net-snmp/sbin/snmpd
```

(14) 安装 Cacti，如下所示。

```
[root@localhost ~]# tar -zxvf cacti-0.8.6j.tar.gz
[root@localhost ~]# mv -r cacti-0.8.6j /usr/local/apache/htdocs/cacti
[root@localhost ~]# vi /usr/local/apache/htdocs/cacti/include/config.php
$database_type = "mysql";
$database_default = "cacti";
$database_hostname = "localhost";
$database_username = "xfimti";
$database_password = "123";
```

(15) 安装 Cactid，如下所示。

```
[root@localhost ~]# tar -zxvf cacti-cactid-0.8.6i.tar.gz
[root@localhost ~]# cd cacti-cactid-0.8.6i
[root@localhost cacti-cactid-0.8.6i]# ./configure
--with-mysql=/usr/local/mysql --with-snmp=/usr/local/net-snmp
[root@localhost cacti-cactid-0.8.6i]# make
[root@localhost cacti-cactid-0.8.6i]# mkdir /usr/local/cactid
[root@localhost cacti-cactid-0.8.6i]# cp cactid cactid.conf /usr/local/cactid
[root@localhost ~]# vi /usr/local/cactid/cactid.conf //修改 cactid 配置文件
DB_Host      127.0.0.1
DB_Database  cacti
DB_User      xfimti
DB_Pass      123
```

(16) 配置 MySQL 数据库，如下所示。

```
mysql> create database cacti;
```

```
Query OK, 1 row affected (0.06 sec)
[root@localhost cacti-0.8.6j]# vi cacti.sql
[root@localhost cacti-0.8.6j]# pwd
/usr/local/apache/htdocs/cacti/cacti-0.8.6j
[root@localhost cacti-0.8.6j]# mysql -u xfimti -p cacti < cacti.sql
Enter password:
```

提示

cacti.sql 脚本直接导入数据库会发生错误,建表语句 TYPE=MyISAM 改为 engine=myisam, 问题就会解决。

(17) 最终完成 cacti 的安装, 在地址栏中输入: <http://192.168.0.100/cacti>, 默认的用户名和密码是 admin, 页面如图 9-6 所示。

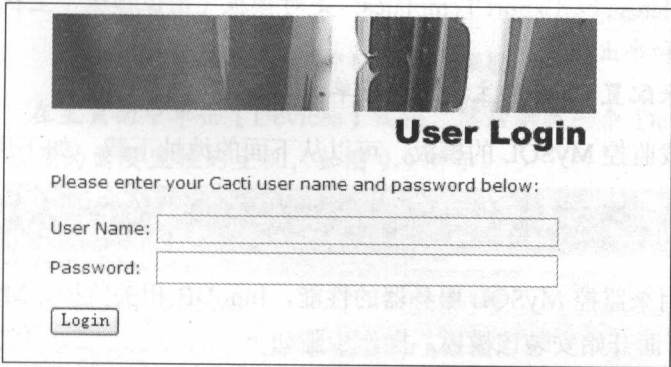


图 9-6 Cacti 登录页面

9.3.2 Cacti 监控 MySQL 服务器

登录 Cacti 后, 主页面如图 9-7 所示, 可以看到两个选项卡【console】和【graphs】。【console】表示控制台, 可以进行所有的配置操作, 【graphs】表示用来查看所有的服务器性能。

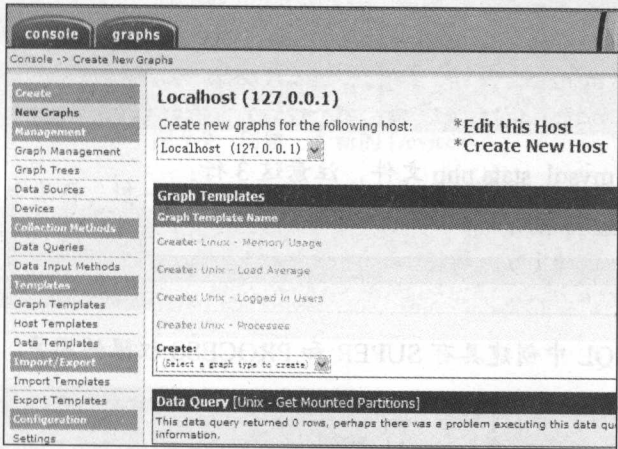


图 9-7 Cacti 登录后的界面

下面先了解下 console 菜单项的具体意义，如下所示。

- New Graphs: 用来创建新图像的快捷方式。
- Graph Management: 可以删除和复制图像，Cacti 会自动创建图像。
- Graph Trees: 在 graphs 界面里，图像或 devices 是树状结果显示的，可以在此设置树的结构。
- Data Sources: 用来管理 rrd 文件。一般无须修改，Cacti 会自己创建 rrd 文件。
- Devices: 可以在此创建新的设备或修改其名称等信息。
- Data Queries 和 Data Input Methods: 用来采集数据的方式。
- Graph Templates、Host Templates 和 Data Templates: 分别表示图像模板、主机类型模板和数据模板。
- Import Templates 和 Export Templates: 是对模板（图像模板、主机类型模板和数据模板）的导入和导出操作。
- Setting: 用来配置 Cacti 的主要配置菜单。

首先，需要下载监控 MySQL 的模板，可以从下面的地址下载，如下所示。

```
http://code.google.com/p/mysql-cacti-templates/downloads/detail?name=better-cacti-templates-1.1.8.tar.gz
```

该模板主要是用来监控 MySQL 服务器的性能，InnoDB 相关监控、MyISAM 相关监控、Nginx 状态监控。下面开始安装该模板，操作步骤如下：

步骤 01 安装 mysql-cacti-templates.tar.gz 文件，如下所示。

```
[root@localhost ~]# tar xzf better-cacti-templates-1.1.8.tar.gz
[root@localhost ~]# cd better-cacti-templates-1.1.8
[root@localhost better-cacti-templates-1.1.8]# cd scripts/
[root@localhost scripts]# ls
ss_get_by_ssh.php  ss_get_mysql_stats.php
[root@localhost scripts]# cp ss_get_mysql_stats.php
/usr/local/apache/htdocs/cacti/scripts/
[root@localhost scripts]# cd /usr/local/apache/htdocs/cacti/scripts/
[root@localhost scripts]# vi ss_get_mysql_stats.php
```

步骤 02 编辑 ss_get_mysql_stats.php 文件，注意这 3 行：

```
$mysql_user = 'xfimti';
$mysql_pass = 'xfimti';
$cache_dir = '/tmp';
```

步骤 03 需要在 MySQL 中创建具有 SUPER 和 PROCESS 权限的用户，如下所示。

```
mysql> grant super,process,select on *.* to xfimti@'%';
Query OK, 0 rows affected (0.00 sec)
```

步骤 04 导入模板。单击管理界面左侧【Import Templates】链接，然后将模板文件 cacti_host_template_x_mysql_server_ht_0.8.6i-server1.1.8.xml 文件导入进去，如图 9-8 所示。

图 9-8 导入监控 MySQL 的模板

步骤 05 添加设备。在主页面中单击【Devices】链接，然后新建一个 Devices 或选择已有的 Devices，设置为需要监控的主机，如图 9-9 所示。

图 9-9 创建一个新的 Devices

步骤 06 设置完成之后保存，过一段时间就可以看见生成的图像了，如图 9-10 所示是部分监控的对象。

| Associated Graph Templates | |
|-------------------------------------------|-------------------------|
| Graph Template Name | Status |
| 1) X InnoDB Active/Locked Transactions GT | Is Being Graphed (Edit) |
| 2) X InnoDB Adaptive Hash Index GT | Is Being Graphed (Edit) |
| 3) X InnoDB Buffer Pool Activity GT | Is Being Graphed (Edit) |
| 4) X InnoDB Buffer Pool GT | Is Being Graphed (Edit) |
| 5) X InnoDB Checkpoint Age GT | Is Being Graphed (Edit) |
| 6) X InnoDB Current Lock Waits GT | Is Being Graphed (Edit) |
| 7) X InnoDB I/O GT | Is Being Graphed (Edit) |
| 8) X InnoDB I/O Pending GT | Is Being Graphed (Edit) |
| 9) X InnoDB Insert Buffer GT | Is Being Graphed (Edit) |

图 9-10 部分 Device 监控的对象

创建设备之后，如果发现一个 SNMP error 的问题，如图 9-11 所示。

MySQL (192.168.0.100)

SNMP Information

SNMP error

*Create Graphs for this Host

Devices [edit: MySQL]

Description

Give this host a meaningful description.

MySQL

Hostname

Fill in the fully qualified hostname for this device.

192.168.0.100

Host Template

Choose what type of host, host template this is. The host template will govern what kinds of data should be gathered from this type of host.

X MySQL Server HT

Disable Host

Check this box to disable all checks for this host.

☐ Disable Host

图 9-11 SNMP error 出错界面

可以通过如下步骤来解决该问题。

步骤 01 确认被检测机器中 snmp 服务是否启动，如果没有启动，可以执行 `service snmpd start` 启动该服务，如下所示。

```
[root@localhost ~]# service snmpd status
snmpd (pid 29856) is running...
```

步骤 02 snmp 服务启动成功后，在检测机器上执行如下命令看能否返回数据。

```
[root@localhost ~]# snmpwalk -v 2c -c public 192.168.0.100 if
Timeout: No Response from 192.168.0.100
```

从结果可以看出，被监控机器的 snmp 服务没有给监控机器授权。

步骤 03 下面需要修改 `/etc/snmp/snmpd.conf` 文件，如下所示。

```
####
# First, map the community name "public" into a "security name"

#      sec.name source          community
com2sec notConfigUser 192.168.0.100 public
```

步骤 04 找到如下代码部分，将 `systemview` 改为 `all`。


```
#      group      context sec.model sec.level prefix read  write notif
access notConfigGroup ""      any      noauth   exact  systemview none none
```

修改之后的代码如下所示。

```
#      group      context sec.model sec.level prefix read  write notif
access notConfigGroup ""      any      noauth   exact  all none none
```

步骤 05 然后找到如下代码部分，将#去掉。

```
##      incl/excl subtree      mask
#view all  included .1      80
```

步骤 06 最后关闭 snmpd 服务，重新启动服务，如下所示。

```
[root@localhost ~]# kill -9 29856
[root@localhost ~]# service snmpd start
Starting snmpd: [ OK ]
```

步骤 07 重新检测机器；执行如下命令，检测能否返回数据。

```
[root@localhost ~]# snmpwalk -v 2c -c public 192.168.0.100 if
.....
IF-MIB::ifOutQLen.2 = Gauge32: 0
IF-MIB::ifOutQLen.3 = Gauge32: 0
IF-MIB::ifSpecific.1 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.2 = OID: SNMPv2-SMI::zeroDotZero
IF-MIB::ifSpecific.3 = OID: SNMPv2-SMI::zeroDotZero
```

步骤 08 执行/usr/local/apache/htdocs/cacti/poller.php 脚本，如下所示。

```
[root@localhost bin]# pwd
/usr/local/php/bin
[root@localhost bin]# ls
pear peardev pecl php php-config phpize
[root@localhost bin]# ./php /usr/local/apache/htdocs/cacti/poller.php
sh: /usr/local/bin/rrdtool: No such file or directory
12/05/2012 12:46:02 AM - POLLER: Poller[0] Maximum runtime of 292 seconds
exceeded. Exiting.
12/05/2012 12:46:02 AM - SYSTEM STATS: Time:292.5174 Method:cmd.php Processes:1
Threads:N/A Hosts:3 HostsPerProcess:3 DataSources:0 RRDsProcessed:0
PHP Warning: pclose(): 48 is not a valid stream resource in
/usr/local/apache/htdocs/cacti/lib/rrd.php on line 47
```

步骤 09 在/usr/local/bin 目录下建立链接 rrdtool 指向 usr/local/rrdtool/bin/rrdtool。

```
[root@localhost bin]# ./php /usr/local/apache/htdocs/cacti/poller.php
12/05/2012 01:43:35 AM - POLLER: Poller[0] Maximum runtime of 292 seconds
exceeded. Exiting.
12/05/2012 01:43:35 AM - SYSTEM STATS: Time:293.0108 Method:cmd.php Processes:1
```

```
Threads:N/A Hosts:3 HostsPerProcess:3 DataSources:0 RRDsProcessed:0
PHP Warning: pclose(): 48 is not a valid stream resource in
/usr/local/apache/htdocs/cacti/lib/rrd.php on line 47
```

步骤 10 下面重新单击 Devices 右侧的 MySQL 链接，如图 9-12 所示。

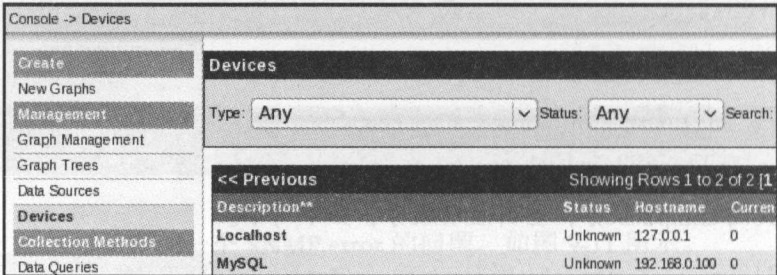


图 9-12 Devices 右侧的 MySQL 链接界面

步骤 11 SNMP error 错误问题被解决了，如图 9-13 所示。

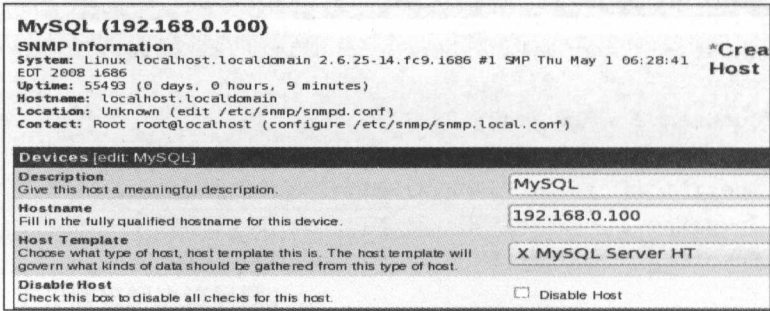


图 9-13 SNMP error 问题成功解决

9.4 小结

MySQL 服务的一些系统检测命令在数据库管理过程中是非常重要的检测方法，本章着重介绍了 Linux 进行使用的系统检查的命令，通过本章的学习可以更好地对 MySQL 数据库以及服务器的性能进行全方位的监控。另外本章重点介绍了监控利器 Nagios，使用 Nagios 监控工具不仅可以监控 MySQL 数据库性能，对数据库集群，复制都能够进行监控。

数据库性能优化中，操作系统以及网络负载、IO、CPU、内存也是制约数据库性能的瓶颈，通过 Nagios 监控工具我们可以更好地监控网络负载、IO、CPU，甚至可以监控磁盘的使用情况。

第 10 章

◀ MySQL Replication ▶

MySQL Replication 是 MySQL 的一个非常重要的功能，主要用于主服务器和从服务器之间的数据复制操作。本章主要讲述了 MySQL Replication 的基本概念、Windows 环境下的复制操作、Linux 环境下的复制操作、如何查看 Slave 的复制进度、日常管理和维护、切换主从服务器的方法等。

10.1 MySQL Replication 概述

MySQL 从 3.25.15 版本开始提供数据库复制 (replication) 功能。MySQL 复制是指从一个 MySQL 主服务器 (master) 将数据复制到另一台或多台 MySQL 从服务器 (slaves) 的过程，将主数据库的 DDL 和 DML 操作通过二进制日志传到复制服务器上，然后在从服务器上对这些日志重新执行，从而使得主从服务器的数据保持同步。

在 MySQL 中，复制操作是异步进行的，slaves 服务器不需要持续地保持连接接收 master 服务器的数据。

MySQL 支持一台主服务器同时向多台从服务器进行复制操作，从服务器同时可以作为其他从服务器的主服务器，如果 MySQL 主服务器访问量比较大，可以通过复制数据，然后在从服务器上进行查询操作，从而降低主服务器的访问压力，同时从服务器作为主服务器的备份，可以避免主服务器因为故障数据丢失的问题。

MySQL 数据库复制操作大致可以分成三个步骤：

- 步骤 01 主服务器将数据的改变记录到二进制日志 (binary log) 中。
- 步骤 02 从服务器将主服务器的 binary log events 复制到它的中继日志 (relay log) 中。
- 步骤 03 从服务器重做中继日志中的事件，将数据的改变与从服务器保持同步。

首先，主服务器会记录二进制日志，每个事务更新数据完成之前，主服务器将这些操作的信息记录在二进制日志里面，在事件写入二进制日志完成后，主服务器通知存储引擎提交事务。

slave 上面的 I/O 进程连接上 Master，并发出日志请求，Master 接收到来自 Slave 的 I/O 进程的请求后，通过负责复制的 I/O 进程根据请求信息读取制定日志指定位置之后的日志信息，返回给 Slave 的 I/O 进程。返回信息中除了日志所包含的信息之外，还包括本次返回的信息已

经到 Master 端的 bin-log 文件的名称以及 bin-log 的位置。

Slave 的 I/O 进程接收到信息后，将接收到的日志内容依次添加到 Slave 端的 relay-log 文件的最末端，并将读取到 Master 端的 bin-log 的文件名和位置记录到 master-info 文件中。

Slave 的 SQL 进程检测到 relay-log 中新增加了内容后，会马上解析 relay-log 的内容成为在 Master 端真实执行时候的那些可执行的内容，并在自身执行。

MySQL 复制环境 90% 以上都是一个 Master 带一个或者多个 Slave 的架构模式。如果 Master 和 Slave 的压力不是太大的话，异步复制的延时一般都很少。尤其是 Slave 端的复制方式改成两个进程处理之后，更是减小了 Slave 端的延时。

提示

对于数据实时性要求不是特别严格的应用，只需要通过廉价的电脑服务器来扩展 Slave 的数量，将读压力分散到多台 Slave 的机器上面，即可解决数据库端的读压力瓶颈。这在很大程度上解决了目前很多中小型网站的数据库压力瓶颈问题，甚至有些大型网站也在使用类似方案解决数据库瓶颈。

10.2 Windows 环境下的 MySQL 主从复制

下面主要通过实验讲解 MySQL Replication 在 Windows 环境下如何配置主从复制的功能。

10.2.1 复制前的准备工作

在 Windows 环境下，如果想实现主从复制功能，需要准备操作环境。本书的操作环境如表 10-1 所示。

表 10-1 MySQL 主从复制所需的环境

| 角色 | IP | 操作系统 | MySQL 版本 |
|--------|---------------|------------|----------------------------------------|
| Master | 192.168.0.208 | Windows XP | mysql-installer-community-5.6.10.1.msi |
| Slave | 192.168.0.206 | Windows XP | mysql-installer-community-5.6.10.1.msi |

提示

读者在做实验的过程中，如果没有很多计算机做实验，可以使用 VMware 虚拟机实现。

10.2.2 Windows 环境下实现主从复制

准备好两台安装 MySQL5.6 的计算机后，即可实现两台 MySQL 服务器主从复制备份操作。具体操作步骤如下。

步骤 01 在 Windows 操作系统下安装好两台主机的 MySQL 服务器，配置好两台主机的 IP 地址，实现两台计算机可以网络连通。

步骤 02 配置 Master 的相关配置信息, 在 Master 主机上开启 binlog 日志, 首先, 看下 datadir 的具体的路径。

```
mysql> show variables like '%datadir%';
```

| Variable_name | Value |
|---------------|------------------------------------------------------|
| datadir | C:\Documents and Settings\...\MySQL Server 5.6\data\ |

1 row in set (0.00 sec)

步骤 03 此时需要打开在 C:\Documents and Settings\All Users\Application Data\MySQL\MySQL Server 5.6 目录下面的配置文件 my.ini, 添加如下代码, 开启 binlog 功能。

```
[mysqld]
log_bin="D:/MySQLlog/binlog"
expire_logs_days = 10
max_binlog_size = 100M
```

提示

此时我们需要在 D 盘下面创建 MySQLlog 文件夹, binlog 日志记录在该文件夹里面, 该配置中其他参数的含义如下所示。

- `expire_logs_days` 表示二进制日志文件删除的天数。
- `max_binlog_size` 表示二进制日志文件最大的大小。

步骤 04 登录 MySQL 之后, 可以执行 `show variables like '%log_bin%'` 命令来测试下 `log_bin` 是否成功开启, 命令语句执行如下。

```
mysql> show variables like '%log_bin%';
```

| Variable_name | Value |
|---------------------------------|--------------------------|
| log_bin | ON |
| log_bin_basename | D:\MySQLlog\binlog |
| log_bin_index | D:\MySQLlog\binlog.index |
| log_bin_trust_function_creators | OFF |
| log_bin_use_v1_row_events | OFF |
| sql_log_bin | ON |

6 rows in set (0.00 sec)

如果 `log_bin` 参数的值为 ON 的话, 那么表示二进制日志文件已经成功开启; 如果为 OFF 的话, 那么表示二进制日志文件开启失败。

步骤 05 在 master 上配置复制所需要的账户, 这里创建一个 repl 的用户, % 表示任何远程地址的 repl 用户都可以连接 master 主机, 语句执行如下所示。

```
mysql> grant replication slave on *.* to repl@'%' identified by '123';
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> flush privileges;
Query OK, 0 rows affected (0.09 sec)
```

步骤 06 在 my.ini 配置文件中配置 Master 主机的相关信息，如下所示。

```
[mysqld]
log_bin="D:/MySQLlog/binlog"
expire_logs_days = 10
max_binlog_size = 100M

server-id = 1
binlog-do-db = test
binlog-ignore-db = mysql
```

这些配置语句的含义如下：

- - server-id 表示服务器标识 id 号，master 和 slave 主机的 server-id 不能一样。
- - binlog-do-db 表示需要复制的数据库，这里以 test 数据库为例。
- - binlog-ignore-db 表示不需要复制的数据库。

步骤 07 重启 Master 主机的 MySQL5.6 服务，然后输入 show master status 命令查询 Master 主机的信息。

```
mysql> show master status \G;
***** 1. row *****
      File: binlog.000003
      Position: 120
      Binlog_Do_DB: test
      Binlog_Ignore_DB: mysql
      Executed_Gtid_Set:
1 row in set (0.00 sec)
```

步骤 08 将 Master 主机的数据备份出来，然后导入到 Slave 主机中去，具体执行语句如下。

```
C:\Program Files\MySQL\MySQL Server 5.6\bin>mysqldump -u root -p -h localhost
test >c:\a.txt
Enter password:
```

将 c:/a.txt 复制到 Slave 主机上面去，然后执行以下操作：

```
C:\Program Files\MySQL\MySQL Server 5.6\bin>mysqldump -u root -p root
-hlocalhost
test <c:\a.txt
Warning: Using a password on the command line interface can be insecure.
-- MySQL dump 10.13 Distrib 5.6.10, for Win32 (x86)
--
```



```
-- Host: localhost      Database: test
--
-----
-- Server version      5.6.10-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0
*/;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2013-04-03 17:25:17
```

步骤 09 配置 Slave 主机 (192.168.0.206)。在 C:\Documents and Settings\All Users\Application Data\MySQL\MySQL Server 5.6 目录下面的配置文件 my.ini 中, 具体配置信息如下所示。

```
[mysql]
default-character-set=utf8
log_bin="D:/MySQLlog/binlog"
expire_logs_days=10
max_binlog_size = 100M

[mysqld]
server-id = 2
```

提示

配置 Slave 主机 my.ini 文件的时候, 需要将 server-id = 2 写到 [mysqld] 后面。另外, 如果配置文件中还有 log_bin 的配置, 可以将它注释掉, 如下所示。

```
# Binary Logging.
# log-bin
# log_bin = "D:/MySQLlog/mysql-bin.log"
```

步骤 10 重启 Slave 主机 (192.168.0.206), 在 Slave 主机 (192.168.0.206) 的 MySQL 中执行

如下命令，关闭 slave 服务，执行如下所示。

```
mysql> stop slave;
Query OK, 0 rows affected (0.05 sec)
```

步骤 11 设置 Slave 从机，实现复制相关的信息。命令执行如下。

```
mysql> change master to
-> master_host='192.168.0.208',
-> master_user='repl',
-> master_password='123',
-> master_log_file='binglog.000003',
-> master_log_pos=120;
Query OK, 0 rows affected, 2 warnings (0.34 sec)
```

各个参数所代表的具体含义如下。

- -master_host 表示实现复制的主机的 IP 地址。
- -master_user 表示实现复制的登录远程主机的用户。
- -master_password 表示实现复制的登录远程主机的密码。
- -master_log_file 表示实现复制的 binlog 日志文件。
- -master_log_pos 表示实现复制的 binlog 日志文件的偏移量。

步骤 12 继续执行操作，显示 Slave 从机的状况，如下所示。

```
mysql> start slave;
Query OK, 0 rows affected (0.11 sec)

mysql> show slave status \G;
***** 1. row *****
      Slave_IO_State:
          Master_Host: 192.168.0.208
          Master_User: repl
          Master_Port: 3306
          Connect_Retry: 60
          Master_Log_File: binglog.000003
          Read_Master_Log_Pos: 120
          Relay_Log_File: 2011-20120220JX-relay-bin.000001
          Relay_Log_Pos: 4
          Relay_Master_Log_File: binglog.000003
          Slave_IO_Running: No
          Slave_SQL_Running: Yes
          Replicate_Do_DB:
          Replicate_Ignore_DB:
          Replicate_Do_Table:
          Replicate_Ignore_Table:
          Replicate_Wild_Do_Table:
```

```

Replicate_Wild_Ignore_Table:
    Last_Errno: 0
    Last_Error:
    Skip_Counter: 0
Exec_Master_Log_Pos: 120
    Relay_Log_Space: 120
    Until_Condition: None
    Until_Log_File:
    Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: NULL
Master_SSL_Verify_Server_Cert: No
    Last_IO_Errno: 1236
    Last_IO_Error: Got fatal error 1236 from master when reading data
a from binary log: 'Could not find first log file name in binary log index file'

    Last_SQL_Errno: 0
    Last_SQL_Error:
Replicate_Ignore_Server_Ids:
    Master_Server_Id: 1
        Master_UUID: a6bd1fa8-8d6b-11e2-97b4-001f3ca9bc3a
    Master_Info_File:C:\Documents and Settings\All Users\Application D
ata\MySQL\MySQL Server 5.6\data\master.info
    SQL_Delay: 0
    SQL_Remaining_Delay: NULL
    Slave_SQL_Running_State: Slave has read all relay log; waiting for the sla
ve I/O thread to update it
    Master_Retry_Count: 86400
    Master_Bind:
    Last_IO_Error_Timestamp: 130403 17:53:15
    Last_SQL_Error_Timestamp:
        Master_SSL_Crl:
        Master_SSL_Crlpath:
    Retrieved_Gtid_Set:
    Executed_Gtid_Set:
        Auto_Position: 0
1 row in set (0.00 sec)

```

在上述执行 `show slave status \G` 命令中很显然存在一些问题，如下所示。

```
Last_IO_Error: Got fatal error 1236 from master when reading dat
```



```
a from binary log: 'Could not find first log file name in binary log index file'
```

下面的步骤是解决该问题的方法，具体操作步骤如下。

步骤 01 重启 Master(192.168.0.208)主机, 执行 `show master status \G` 命令, 记下 File 和 Position 的值, 后面 Slave 主机可能会用到。命令执行如下:

```
mysql> show master status \G;
***** 1. row *****
      File: binlog.000004
      Position: 120
      Binlog_Do_DB: test
      Binlog_Ignore_DB: mysql
      Executed_Gtid_Set:
1 row in set (0.00 sec)
```

步骤 02 在 Slave (192.168.0.206) 主机上重新设置信息, 命令执行如下所示。

```
mysql> stop slave;
Query OK, 0 rows affected (0.01 sec)

mysql> change master to
      -> master_log_file='binlog.000004',
      -> master_log_pos = 120;
Query OK, 0 rows affected (0.16 sec)

mysql> start slave;
Query OK, 0 rows affected (0.05 sec)

mysql> show slave status\G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 192.168.0.208
      Master_User: repl
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: binlog.000004
      Read_Master_Log_Pos: 120
      Relay_Log_File: 2011-20120220JX-relay-bin.000002
      Relay_Log_Pos: 280
      Relay_Master_Log_File: binlog.000004
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
```

```

Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
    Last_Errno: 0
    Last_Error:
    Skip_Counter: 0
Exec_Master_Log_Pos: 120
Relay_Log_Space: 463
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
    Last_IO_Errno: 0
    Last_IO_Error:
    Last_SQL_Errno: 0
    Last_SQL_Error:
Replicate_Ignore_Server_Ids:
    Master_Server_Id: 1
        Master_UUID: a6bd1fa8-8d6b-11e2-97b4-001f3ca9bc3a
    Master_Info_File: C:\Documents and Settings\All Users\Application D
ata\MySQL\MySQL Server 5.6\data\master.info
    SQL_Delay: 0
    SQL_Remaining_Delay: NULL
    Slave_SQL_Running_State: Slave has read all relay log; waiting for the sla
ve I/O thread to update it
    Master_Retry_Count: 86400
    Master_Bind:
    Last_IO_Error_Timestamp:
    Last_SQL_Error_Timestamp:
    Master_SSL_Crl:
    Master_SSL_Crlpath:
    Retrieved_Gtid_Set:
    Executed_Gtid_Set:
    Auto_Position: 0
1 row in set (0.00 sec)

```

由此可见，问题完全解决，接下来可以进行 Window 环境下主从复制的测试。

10.2.3 Windows 环境下主从复制测试

在 Windows 环境中测试主从复制操作。具体操作步骤如下。

步骤 01 在 Master 主机的 MySQL 环境下，执行如下命令。

```
mysql> use test;
Database changed
mysql> create table rep_test(
-> data integer
-> );
Query OK, 0 rows affected (0.02 sec)

mysql> insert into rep_test values(2);
Query OK, 1 row affected (0.06 sec)
```

步骤 02 在 Slave 主机的 MySQL 环境下，查看主机刚才添加的表和数据是否成功同步到从机上，命令执行如下所示。

```
mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| t2              |
+-----+
1 row in set (0.00 sec)

mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| rep_test       |
| t2             |
+-----+
2 rows in set (0.00 sec)

mysql> select *from rep_test;
+-----+
| data |
+-----+
| 2    |
+-----+
1 row in set (0.02 sec)
```


测试表明数据已经成功地同步到 Slave 主机上了, 实验中只是用到主从同步, 在实际生产环境中 MySQL 架构可能会用到一主多从的架构, 这里不再叙述。

10.3 Linux 环境下的 MySQL 复制

在现实的生产环境中单机实现的主从复制比较少, 通常会使用一主多从的架构体系, 本节为了读者朋友更好地实现本机主从复制, 需要在 Linux 环境下面通过 `mysqld_multi` 实现单机的主从复制。本章节使用的是 Fedora 操作系统。

10.3.1 下载并安装 MySQL 5.6

很多熟悉 MySQL 的用户都喜欢使用源码包来进行安装, 因为在安装源码的过程中可以非常方便地进行性能的优化, 下面就源码安装过程中涉及的优化项进行简单介绍。

步骤 01 先下载 `mysql-5.6.10.tar.gz` 源文件。读者可以在下载页面 <http://dev.mysql.com/downloads/mysql/5.6.html#downloads> 选择【Source Code】平台, 然后选择下载 `mysql-5.6.10.tar.gz` 源码, 如图 10-1 所示。

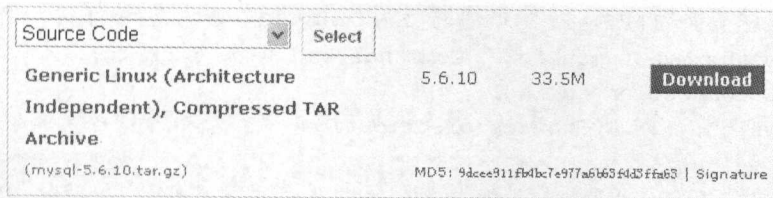


图 10-1 MySQL 源码下载

步骤 02 下载完 `mysql-5.6.10.tar.gz` 后, 创建 MySQL 安装程序的目录和数据文件的目录。命令执行如下。

```
[root@localhost ~]# mkdir -p /usr/local/mysql
[root@localhost ~]# mkdir -p /usr/local/mysql/data
[root@localhost ~]# groupadd mysql
[root@localhost ~]# useradd -r -g mysql mysql
```

步骤 03 解压缩 MySQL 源代码, 这里使用的是 `cmake2.8.4` 来编译 MySQL 源代码。命令执行如下。

```
[root@localhost ~]# groupadd mysql
[root@localhost ~]# useradd -r -g mysql mysql
[root@localhost ~]# tar -zxvf mysql-5.6.10.tar.gz
[root@localhost ~]# cd mysql-5.6.10
[root@localhost mysql-5.6.10]# cmake .
```

```
[root@localhost mysql-5.6.10]# make && make install
```

步骤 04 修改 MySQL 安装程序的目录权限。命令执行如下。

```
[root@localhost ~]# chown -R mysql:mysql /usr/local/mysql
```

步骤 05 安装 MySQL 5.6 的源码。命令执行如下。

```
[root@localhost ~]# cd /usr/local/mysql/scripts
[root@localhost scripts]# ./mysql_install_db --user=mysql
--basedir=/usr/local/mysql --datadir=/usr/local/mysql/data
[root@localhost ~]# cd /usr/local/mysql/support-files
[root@localhost support-files]# cp mysql.server /etc/rc.d/init.d/mysql
[root@localhost support-files]# cp my-default.cnf /etc/my.cnf
[root@localhost ~]# chkconfig --add mysql
[root@localhost ~]# chkconfig mysql on
```

步骤 06 启动 MySQL 5.6 服务。命令执行如下。

```
[root@localhost ~]# service mysql start
Starting MySQL...
```

[OK]

步骤 07 在 Fedora 操作系统中登录 MySQL 5.6，默认用户 root，密码为空，命令执行如下所示。

```
[root@localhost ~]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.10 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

10.3.2 单机主从复制前的准备工作

MySQL 服务器可以采用主从机制进行备份，如果一对一进行备份对于生成环境而言比较浪费资源，主服务器把数据变化记录到主日志，然后从服务器通过 I/O 线程读取主服务器的日志，并将它写入到从服务器的中继日志中，接着 SQL 线程读取中继日志，并且在从服务器上重放，从而实现 MySQL 复制，具体如图 10-2 所示。

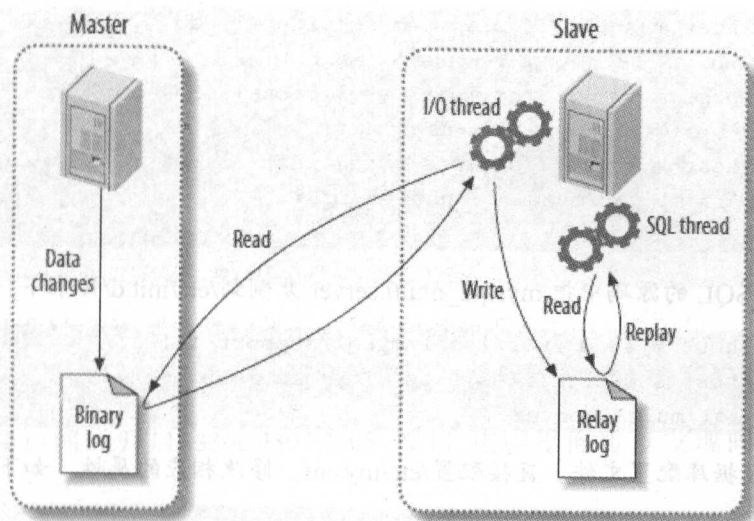


图 10-2 MySQL 复制

MySQL 具有可以运行多个实例的功能，这个功能是通过 `mysqld_multi` 实现的。当一台机器上需要运行多个 MySQL 服务器的时候，`mysqld_multi` 是管理多个 `mysqld` 的服务进程，这些服务进程用不同的 `unix socket` 或是监听不同的端口，通过命令，可以启动、关闭和报告所管理的服务器的状态。

下面介绍在一台服务器上使用 `mysqld_multi` 管理多个 MySQL 服务进程。具体操作步骤如下。

步骤 01 初始化多个实例数据库，首先要停止掉 MySQL 服务器。命令执行如下。

```
[root@localhost ~]# service mysql stop
Shutting down MySQL.
```

[OK]

提示

此时可以采用 `netstat` 命令查看 3306 关闭了没有，如果没有查询出结果，那么说明 MySQL 服务器已经成功地关闭掉了。

步骤 02 把常用到的工具添加到 `/usr/bin` 目录，命令执行如下。

```
[root@localhost~]# ln -s /usr/local/mysql/bin/mysqld_multi
/usr/bin/mysqld_multi
[root@localhost~]# ln -s /usr/local/mysql/scripts/mysql_install_db
/usr/bin/mysql_install_db
```

步骤 03 初始化 3 个数据目录并安装 3 个 `mysql` 服务，命令执行如下。

```
[root@localhost ~]# cd /usr/local/mysql/
[root@localhost mysql]# mkdir -p /usr/local/var/mysql1
[root@localhost mysql]# mkdir -p /usr/local/var/mysql2
[root@localhost mysql]# mkdir -p /usr/local/var/mysql3
```



```
[root@localhost mysql]# ./scripts/mysql_install_db
--datadir=/usr/local/var/mysql1 --user=mysql
[root@localhost mysql]# ./scripts/mysql_install_db
--datadir=/usr/local/var/mysql2 --user=mysql
[root@localhost mysql]# ./scripts/mysql_install_db
--datadir=/usr/local/var/mysql3 --user=mysql
```

步骤 04 从 MySQL 的源码中把 `mysqld_multi.server` 复制到 `/etc/init.d/` 目录下，命令执行如下。

```
[root@localhost ~]# cd /usr/local/mysql/support-files/
[root@localhost support-files]# cp ./mysqld_multi.server
/etc/init.d/mysqld_multi.server
```

步骤 05 配置数据库配置文件。直接配置 `/etc/my.cnf`，修改相应的属性，如下所示。

```
# The MySQL server
[mysqld_multi]
mysqld      = /usr/local/mysql/bin/mysqld_safe
mysqladmin  = /usr/local/mysql/bin/mysqladmin
user        = root

[mysqld1]
port        = 3306

[mysqld2]
port        = 3307
socket      = /temp/mysql2.sock
datadir     = /usr/local/var/mysql2

[mysqld3]
port        = 3308
socket      = /temp/mysql3.sock
datadir     = /usr/local/var/mysql3

[mysqld]
```

步骤 06 查看数据库的状态。命令执行如下。

```
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf report
Reporting MySQL servers
MySQL server from group: mysqld1 is not running
MySQL server from group: mysqld2 is not running
MySQL server from group: mysqld3 is not running
```

此时，发现 MySQL 服务器的状态，不能够打开所需的文件，程序发生错误。

步骤 07 使用 `mysqld_multi` 启动 MySQL 服务器，命令执行如下。

```
[root@localhost mysql1]# mysqld_multi --defaults-extra-file=/etc/my.cnf stop
[root@localhost mysql1]# mysqld_multi --defaults-extra-file=/etc/my.cnf start
[root@localhost mysql1]# mysqld_multi --defaults-extra-file=/etc/my.cnf
report
Reporting MySQL servers
MySQL server from group: mysqld1 is running
MySQL server from group: mysqld2 is running
MySQL server from group: mysqld3 is running
```

步骤 08 下面测试 MySQL 服务器的状态，命令执行如下。

```
[root@localhost ~]# netstat -an|grep 330
```

此时发现端口同时开启 3306,3307,3308 端口,进程里面可以发现同时开启了两个 mysql_safe 进程。

步骤 09 下面登录查看 MySQL 数据库，命令执行如下。

```
[root@localhost data]# mysql -u root -p -P 3306
[root@localhost data]# mysql -u root -p -P 3307
[root@localhost data]# mysql -u root -p -P 3308
```

此时可以顺利登录到数据库，此时也可以通过 ps 命令发现后台产生了 3 个 mysqld 进程的实例。

步骤 10 直接登录 MySQL 服务器，执行 show variables 命令发现三个 MySQL 服务器的 pid_file, socket 参数都一样，命令执行如下。

```
mysql> show variables like 'socket';
```

| Variable_name | Value |
|---------------|-----------------|
| socket | /tmp/mysql.sock |

1 row in set (0.00 sec)

```
mysql> show variables like 'pid%';
```

| Variable_name | Value |
|---------------|----------------------------------|
| pid_file | /usr/local/var/mysql1/mysql1.pid |

1 row in set (0.00 sec)

此时，通过登录 MySQL 服务器自带参数，可以解决以上的问题。命令执行如下所示。

```
[root@localhost ~]# mysql -u root -S /tmp/mysql2.sock
Welcome to the MySQL monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 1
Server version: 5.0.89 MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| mysql              |
| test               |
+-----+
3 rows in set (0.04 sec)

mysql> show variables like 'pid%';
+-----+-----+
| Variable_name | Value                                     |
+-----+-----+
| pid_file      | /usr/local/var/mysql2/mysql2.pid      |
+-----+-----+
1 row in set (0.00 sec)
```

测试结果可知，问题已经解决了，接下来启动 3 个数据库，可以直接使用了。

```
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf stop 1-3
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf star 1-3
```

10.3.3 mysqld_multi 实现单机主从复制

MySQL 的复制至少需要两个 MySQL 服务，这些 MySQL 服务可以分布在不同的服务器上，也可以在一台服务器上启动多个服务。

MySQL 的复制 (Replication) 是一个异步的复制过程，从 Master 复制到 Slave。从 Master 和 Slave 实现整个复制过程由三个线程完成，其中两个线程 (SQL 线程和 IO 线程) 在 Slave 端，另一个线程 (IO 线程) 在 Master 端。要实现复制过程，Master 必须打开 Binary Log 功能，复制过程其实就是 Slave 从 Master 端获取 bin 日志，然后在自己服务器上完全顺序执行日志中所记录的各种操作。

在 Fedora 操作系统中使用 Mysqld_multi 单机实现主从复制的具体配置如表 10-2 所示。

表 10-2 Mysqld_multi 单机实现主从复制的具体配置环境

| 角色 | IP | 操作系统 | MySQL 版本 | 端口 |
|--------|---------------|----------|---------------------|------|
| Master | 192.168.0.208 | Fedora14 | mysql-5.6.10.tar.gz | 3306 |
| Slave | 192.168.0.208 | Fedora14 | mysql-5.6.10.tar.gz | 3307 |
| Slave | 192.168.0.208 | Fedora14 | mysql-5.6.10.tar.gz | 3308 |

下面是采用 `mysqld_multi` 实现单机 MySQL 服务器主从复制。具体操作步骤如下。

步骤 01 使用 `mysqld_multi` 开启上一节已经设定好的三个 MySQL 服务，命令执行如下。

```
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf start 1-3
[root@localhost ~]# netstat -an|grep 330
tcp    0      0 0.0.0.0:3306          0.0.0.0:*             LISTEN
tcp    0      0 0.0.0.0:3307          0.0.0.0:*             LISTEN
tcp    0      0 0.0.0.0:3308          0.0.0.0:*             LISTEN
```

步骤 02 登录 Master 主服务器，设置一个复制使用的账户，并授予 `REPLICATION SLAVE` 权限。这里创建一个复制用户 `repl`。

```
[root@localhost ~]# mysql -u root -p -P 3306
Enter password:

mysql> grant replication slave on *.* to 'repl'@'localhost' identified by '123';
Query OK, 0 rows affected (0.04 sec)

mysql> grant replication slave on *.* to 'repl'@'%' identified by '123';
Query OK, 0 rows affected (0.04 sec)
```

步骤 03 修改 Master 主数据库服务器的配置文件 `my.cnf`，开启 `BINLOG`，并设置 `server-id` 的值。需要重启服务器之后才生效。

```
[root@localhost ~]# vi /etc/my.cnf
[mysqld]
port      = 3306
log-bin    = /usr/local/var/mysql1/mysql-bin
server-id  = 1

[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf stop 1-3
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf start 1-3
```

步骤 04 在 Master 主服务器上，设置度锁定有效，这个操作为了确保没有数据库操作，以便获得一致性的快照。

```
[root@localhost ~]# mysql -u root -P 3306 -S /tmp/mysql.sock
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.10-log Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> flush tables with read lock;
Query OK, 0 rows affected (0.01 sec)
```

步骤 05 用 `show master status` 命令查看日志情况，查询得到主服务器上当前的二进制日志名和偏移量值。这个操作的目的是为了从数据库启动以后，从这个点开始进行数据的恢复。

```
mysql> show master status \G;
***** 1. row *****
      File: mysql-bin.000001
      Position: 120
      Binlog_Do_DB:
      Binlog_Ignore_DB:
      Executed_Gtid_Set:
1 row in set (0.00 sec)
```

步骤 06 主数据库服务此时可以做一个备份，可以在服务器停止的情况下直接使用系统复制命令。

```
[root@localhost mysql1]#tar -cvf data.tar data
```

步骤 07 主数据库备份完成后，主数据库恢复写操作，命令执行如下。

```
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

提示

Master 主服务器的配置已经成功，如果 `my.cnf` 的 `mysqld` 选项设置 `server-id` 参数，但从服务器没有设置 `server-id`，那么启动从服务器会发生错误如下。

```
mysql> start slave;
ERROR 1200 (HY000): The server is not configured as slave; fix in config file
or with CHANGE MASTER TO
```

步骤 08 接下来继续编辑 `/etc/my.cnf` 文件，具体配置项如下。

```
# The MySQL server
[mysqld_multi]
mysqld      = /usr/local/mysql/bin/mysqld_safe
mysqladmin  = /usr/local/mysql/bin/mysqladmin
user        = root

[mysqld1]
port        = 3306
log-bin     = /usr/local/var/mysql1/mysql-bin
```

```

server-id      = 1

[mysqld2]
port          = 3307
socket        = /temp/mysql2.sock
datadir       = /usr/local/var/mysql2
log-bin       = /usr/local/var/mysql2/mysql-bin
server-id     = 2

[mysqld3]
port          = 3308
socket        = /temp/mysql3.sock
datadir       = /usr/local/var/mysql3
log-bin       = /usr/local/var/mysql3/mysql-bin
server-id     = 3

[mysqld]

```

步骤 09 重启 Master 主服务器，命令执行如下。

```

[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf stop 1-3
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf start 1-3
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf report
Reporting MySQL servers
MySQL server from group: mysqld1 is running
MySQL server from group: mysqld2 is running
MySQL server from group: mysqld3 is running
[root@localhost ~]#

```

步骤 10 对从数据库服务器做相应设置，此时需要制定复制使用的用户，主数据的 IP 地址，端口以及开始复制的日志文件和位置等，具体设置如下。

```

[root@localhost ~]# mysql -u root -p -P 3307 -S /temp/mysql2.sock
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.10-log Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show variables like '%log_bin%';

```



```

+-----+-----+
| Variable_name          | Value                               |
+-----+-----+
| log_bin                | ON                                 |
| log_bin_basename       | /usr/local/var/mysql2/mysql-bin   |
| log_bin_index          | /usr/local/var/mysql2/mysql-bin.index |
| log_bin_trust_function_creators | OFF                               |
| log_bin_use_vl_row_events | OFF                               |
| sql_log_bin            | ON                                 |
+-----+-----+
6 rows in set (0.01 sec)

```

```

mysql> stop slave;
Query OK, 0 rows affected, 1 warning (0.00 sec)

```

```

mysql> change master to
-> master_host='127.0.0.1',
-> master_user='repl',
-> master_password='123',
-> master_log_file='mysql-bin.000001',
-> master_log_pos=120;
Query OK, 0 rows affected, 2 warnings (0.10 sec)

```

步骤 11 在从服务器上执行 `show slave status\G` 命令查询从服务器的状态，命令执行如下。

```

mysql> start slave;
Query OK, 0 rows affected (0.12 sec)

mysql> show slave status \G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 127.0.0.1
      Master_User: repl
      Master_Port: 3306
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000002
      Read_Master_Log_Pos: 120
      Relay_Log_File: localhost-relay-bin.000003
      Relay_Log_Pos: 283
      Relay_Master_Log_File: mysql-bin.000002
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:

```

```

Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
    Last_Errno: 0
    Last_Error:
    Skip_Counter: 0
Exec_Master_Log_Pos: 120
Relay_Log_Space: 623
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
    Last_IO_Errno: 0
    Last_IO_Error:
    Last_SQL_Errno: 0
    Last_SQL_Error:
Replicate_Ignore_Server_Ids:
    Master_Server_Id: 1
        Master_UUID: 5c2dfa9a-8327-11e2-94c2-000c296d88f8
    Master_Info_File: /usr/local/var/mysql2/master.info
    SQL_Delay: 0
    SQL_Remaining_Delay: NULL
Slave_SQL_Running_State: Slave has read all relay log; waiting for the slave
I/O thread to update it
    Master_Retry_Count: 86400
    Master_Bind:
    Last_IO_Error_Timestamp:
    Last_SQL_Error_Timestamp:
    Master_SSL_Crl:
    Master_SSL_Crlpath:
    Retrieved_Gtid_Set:
    Executed_Gtid_Set:
    Auto_Position: 0
1 row in set (0.00 sec)

```

步骤 12 此时发现从服务器已经成功设置，此时也可以执行 `show processlist \G` 命令查询从服务器的进程状态，命令执行如下。


```
mysql> show processlist \G;
***** 1. row *****
    Id: 7
   User: root
  Host: localhost
    db: NULL
Command: Query
   Time: 0
  State: init
   Info: show processlist
***** 2. row *****
    Id: 10
   User: repl
  Host: localhost:45056
    db: NULL
Command: Binlog Dump
   Time: 202
  State: Master has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
2 rows in set (0.00 sec)
```

结果表明 slave 已经连接上 master，并开始接受并执行日志。

步骤 13 此时可以测试复制服务的正确性，在 Master 主数据库上执行一个更新操作，观察是否在从服务器上同步。下面在主数据库的 test 库上创建一个测试表，然后插入数据，命令执行如下。

```
[root@localhost ~]# mysql -u root -p -P 3306 -S /tmp/mysql.sock
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 5.6.10-log Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test;
Database changed
mysql> show tables;
Empty set (0.00 sec)
```



```
mysql> create table repl_test(id int);
Query OK, 0 rows affected (0.18 sec)

mysql> insert into repl_test values(1),(2);
Query OK, 2 rows affected (0.06 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

步骤 14 在从服务器上检测新的表是否被创建，数据是否同步，执行命令如下。

```
[root@localhost ~]# mysql -u root -P 3307 -S /temp/mysql2.sock
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.6.10-log Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| repl_test      |
+-----+
1 row in set (0.00 sec)

mysql> select * from repl_test;
+-----+
| id |
+-----+
| 1 |
| 2 |
+-----+
2 rows in set (0.01 sec)

mysql>
```

从结果可以看出，端口为 3306 的 Master 主机上的数据已经可以正确地同步到端口为 3307 的 Slave 主机的数据库上，复制服务配置成功完成。另外一个端口为 3308 的从机的配置跟端口为 3307 的一样操作，这里不再重复叙述。

10.3.4 不同服务器之间实现主从复制

在大多数情况下,采用不同的 MySQL 主从复制比较常见,不同 IP 地址的服务器上的 MySQL 服务器实现一对一复制跟上一节比较相似,具体的配置步骤如下。

步骤 01 确保主从服务器上安装了相同版本的数据库,设定主服务器的 IP 是 192.168.1.100,从服务器的 IP 是 192.168.1.101。

步骤 02 登录主服务器,设置一个复制使用的账户,并授予 REPLICATION SLAVE 权限。这里创建一个复制用户 repl。

```
mysql> grant replication slave on *.* to 'repl'@'192.168.1.101' identified by
'123';
Query OK, 0 rows affected (0.00 sec)
```

步骤 03 修改主数据库服务器的配置文件 my.cnf,开启 BINLOG,并设置 server-id 的值。需要重启服务器之后才生效。

```
my.cnf 中修改
[mysqld]
log-bin = /usr/local/var/mysql1/mysql-bin
server-id = 1
```

步骤 04 在主服务器上,设置读锁定有效,这个操作为了确保没有数据库操作,以便获得一致性的快照。

```
mysql> flush tables with read lock;
Query OK, 0 rows affected (0.00 sec)
```

步骤 05 查询主服务器上当前的二进制日志名和偏移值。这个操作的目的是为了在从数据库启动以后,从这个点进行数据库的恢复。

```
mysql> show master status;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000029 |      109 |              |                  |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

步骤 06 主数据库停止更新操作,需要生成数据库的备份,可以通过 mysqldump 导出数据或者使用 ibbackup 工具进行数据库的备份,如果主数据库停止,那么可以直接使用 cp 复制全部数据文件到从数据库服务器上。

主数据库备份完成后,主数据库恢复写操作,命令执行如下。

```
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
```

步骤 07 修改从数据库的配置文件 my.cnf，增加 server-id 参数。Server-id 的值是唯一的，不能和主数据库的配置相同，如果有多个从数据库，每个从数据库都必须有自己唯一的 server-id 值。

```
my.cnf
[mysqld]
Server-id = 2
```

步骤 08 启动从数据库。

```
[root@localhost ~]# mysqld_safe --skip-slave-start &
```

步骤 09 对从数据库服务器做相应的设置，指定复制使用的用户、主数据库服务器的 IP、端口以及开始执行复制的日志文件和位置，命令执行如下。

```
mysql> stop slave;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> change master to
-> master_host='192.168.1.100',
-> master_user='repl',
-> master_password='123',
-> master_log_file='mysql-bin.000029',
-> master_log_pos=109;
Query OK, 0 rows affected (0.00 sec)
```

步骤 10 在从服务器上，启动 slave 线程。

```
mysql> start slave;
Query OK, 0 rows affected (0.00 sec)
```

步骤 11 在从服务器上执行 show slave status\G 命令查询从服务器的状态。

```
mysql> show slave status\G;
```

此时也可以执行 show processlist \G 命令查询从服务器的进程状态。

```
mysql> show processlist \G;
```

接下来可以测试复制服务的正确性，在主数据库上执行一个更新操作，观察是否在从数据库上同步。具体方法与上一节相似，这里不再重复讲述。

10.3.5 MySQL 主要复制启动选项

MySQL 安装配置的时候，已经介绍了几个启动时的常用参数，其中包括 MASTER_HOST、MASTER_PORT、MASTER_USER、MASTER_PASSWORD、MASTER_LOG_FILE 和 MASTER_LOG_POS。这几个参数需要在从服务器上配置，下面介绍几个常用的启动选项，如 log-slave-updates、master-connect-retry、read-only 和 slave-skip-errors 等。

(1) log-slave-updates

`log-slave-updates` 参数主要用来配置从服务器的更新是否写入二进制日志，该选项默认是不打开的，如果这个从服务器同时也作为其他服务器的主服务器，搭建一个链式的复制，那么就需要开启这个选项，这样他的从服务器才能获取它的二进制日志进行同步操作。

(2) master-connect-retry

`master-connect-retry` 参数是用来设置在和主服务器连接丢失的时候，重试的时间间隔，默认是 60 秒。

(3) read-only

`read-only` 是用来限制普通用户对从数据库的更新操作，以确保从数据库的安全性，不过如果是超级用户依然可以对从数据库进行更新操作。如果主数据库创建了一个普通用户，在默认情况下，该用户是可以更新从数据库中的数据的，如果使用 `read-only` 选项启动从数据库以后，该用户对从数据库的更新会提示错误。

使用 `read-only` 选项启动语法如下。

```
[root@localhost ~]#mysqld_safe -read-only&
```

(4) slave-skip-errors

在复制的过程中，从服务器可能会执行 BINLOG 中的错误的 SQL 语句，此时如果不忽略错误，从服务器将会停止复制进程，等待用户处理错误。这种错误如果不能及时发现，将会对应用或者备份产生影响。`slave-skip-errors` 的作用就是用来定义复制过程中从服务器可以自动跳过的错误号，设置该参数后，MySQL 会自动跳过所配置的一系列错误，直接执行后面的 SQL 语句，该参数可以定义多个错误号，如果设置成 `all`，则表示跳过所有的错误，具体语法如下。

```
vi /etc/my.cnf
slave-skip-errors=1007,1051,1062
```

如果从数据库主要是作为主数据库的备份，那么就不应该使用这个启动参数，设置不当，很可能造成主从数据库的数据不同步。如果从数据库仅仅是为了分担主数据库的查询压力，并且对数据的完整性要求不是很严格，那么这个选项可以减轻数据库管理员维护从数据库的工作量。

10.3.6 指定复制的数据库或者表

MySQL 数据库可以指定需要复制到从数据库上的数据库或者表，有时候用户只需要将主数据库中的某些关键表复制到从服务器上，或者只需要将某些提供查询的表复制到主数据库上，经常可以使用 `replicate-do-db`、`replicate-do-table`、`replicate-ignore-db`、`replicate-ignore-table` 或 `replicate-while-do-table` 指定复制的数据或者表。

1. replicate-do-table 和 replicate-ignore-table 的用法

步骤 01 启动主从数据库，首先在主数据库 `test` 库中创建两个表，`rep_t1` 和 `rep_t2` 表。

```
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf start 1-3
[root@localhost ~]# mysql -u root -p -P 3306 -S /tmp/mysql.sock
```

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 11

Server version: 5.6.10-log Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
mysql> show databases;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| cc                 |
| mysql              |
| test               |
| tt                 |
+-----+
```

5 rows in set (0.11 sec)

```
mysql> show tables;
```

```
+-----+
| Tables_in_test    |
+-----+
| rep_t1             |
| rep_t2             |
+-----+
```

2 rows in set (0.00 sec)

```
mysql> select * from rep_t1;
```

```
+-----+
| data |
+-----+
| 1    |
| 2    |
| 3    |
+-----+
```

3 rows in set (0.05 sec)

```
mysql> select * from rep_t2;
Empty set (0.04 sec)
```

步骤 02 关闭数据库服务器，编辑从数据库配置参数 `replicate-do-table=test.rep_t1` 指定 test 数据库中的 `rep_t1` 表被复制，`replicate-ignore-table=test.rep_t2` 指定 test 库中的 `rep_t2` 表不会被复制数据。

```
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf stop 1-3

Vi /etc/my.cnf
[mysqld2]
...
port      = 3307
socket    = /temp/mysql2.sock
server-id = 2
replicate-do-table=test.rep_t1
replicate-ignore-table=test.rep_t2
```

步骤 03 启动主从服务器，命令执行如下。

```
mysql> start slave;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> show slave status\G;
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
        Master_Host: 127.0.0.1
        Master_User: repl
        Master_Port: 3306
        Connect_Retry: 60
        Master_Log_File: mysql-bin.000013
        Read_Master_Log_Pos: 98
        Relay_Log_File: mysql2-relay-bin.000035
        Relay_Log_Pos: 235
        Relay_Master_Log_File: mysql-bin.000013
        Slave_IO_Running: Yes
        Slave_SQL_Running: Yes
        Replicate_Do_DB:
        Replicate_Ignore_DB:
        Replicate_Do_Table: test.rep_t1,test.rep_t1
        Replicate_Ignore_Table: test.rep_t2,test.rep_t2
        Replicate_Wild_Do_Table:
        Replicate_Wild_Ignore_Table:
        Last_Errno: 0
        Last_Error:
```



```

Skip_Counter: 0
Exec_Master_Log_Pos: 98
Relay_Log_Space: 235
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
1 row in set (0.00 sec)

```

步骤 04 主从服务器都成功启动后, 下面开始更新主数据库 test 库中的 rep_t1 表和 rep_t2 表, 修改主数据库中的数据如下。

```

mysql> insert into rep_t1 values(888);
Query OK, 1 row affected (0.00 sec)

```

```

mysql> insert into rep_t2 values(888);
Query OK, 1 row affected (0.01 sec)

```

步骤 05 登录从数据库, 查询 test 库中的表 rep_t1 和 rep_t2 的数据更新的情况, 具体查询语句如下。

```

[root@localhost ~]# mysql -u root -P 3307 -S /temp/mysql2.sock

```

```

Enter password:

```

```

Welcome to the MySQL monitor.  Commands end with ; or \g.

```

```

Your MySQL connection id is 11

```

```

Server version: 5.6.10-log Source distribution

```

```

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

```

```

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

```

```

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```

```

mysql> use test;

```

```

Database changed

```

```

mysql> show tables;

```

```

+-----+
| Tables_in_test |

```

```

+-----+
| rep_t1 |
| rep_t2 |
+-----+
2 rows in set (0.00 sec)

mysql> select * from rep_t1;
+-----+
| data |
+-----+
| 1 |
| 2 |
| 3 |
| 888 |
+-----+
4 rows in set (0.00 sec)

mysql> select * from rep_t2;
Empty set (0.00 sec)

```

从测试的结果可以看到，主表中的 rep_t1 的数据已经复制到从服务器上了，而 rep_t2 中的数据没有被复制。

2. replicate-do-db 和 replicate-ignore-db 的用法

步骤 01 启动主从数据库服务器，查询主数据库中的主要有哪些数据库，命令执行如下。

```

[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf start 1-3

[root@localhost ~]# mysql -u root -P 3306 -S /tmp/mysql.sock
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 5.6.10-log Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+

```

```
| Database |
+-----+
| information_schema |
| cc |
| mysql |
| test |
| tt |
+-----+
5 rows in set (0.00 sec)
```

```
mysql>
```

步骤 02 使用 `mysqldump` 工具将主数据库中的所有信息导出到 `all.sql` 脚本文件中。

```
[root@localhost ~]# mysqldump -u root -P 3306 -S /tmp/mysql.sock --all-databases
>all.sql
```

步骤 03 登录从数据库，导入 `all.sql` 中的数据，保持从服务器与主数据库数据一致。

```
[root@localhost ~]# mysql -u root -P 3307 -S /temp/mysql2.sock
```

```
Enter password:
```

```
Welcome to the MySQL monitor. Commands end with ; or \g.
```

```
Your MySQL connection id is 11
```

```
Server version: 5.6.10-log Source distribution
```

```
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
```

```
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| test |
+-----+
3 rows in set (0.00 sec)
```

```
mysql> source ./all.sql
```

```
Query OK, 0 rows affected (0.05 sec)
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
...
```



```
mysql> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| cc                 |
| mysql              |
| test               |
| tt                 |
+-----+
5 rows in set (0.00 sec)
```

步骤 04 关闭从数据库，然后编辑数据库的配置文件，`replicate-do-db` 表示从服务器可以复制的数据库的名字，如果有多个数据库，那么可以重复写多个 `replicate-do-db` 配置，`replicate-ignore-db` 表示从服务器复制过程中忽略复制该配置设置的数据库名称。

```
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf stop 1-3

vi /etc/my.cnf
[mysqld2]
....
port      = 3307
socket     = /tmp/mysql2.sock
server-id = 2
#replicate-do-table=test.rep_t1
#replicate-ignore-table=test.rep_t2
replicate-do-db=test
replicate-do-db=cc
replicate-ignore-db=tt
```

步骤 05 启动主从数据库，然后在主数据库 `cc` 库中增加表 `cc_t1` 表，在 `tt` 库中增加表 `tt_t1` 表。

```
[root@localhost ~]# mysqld_multi --defaults-extra-file=/etc/my.cnf start 1-3

mysql> use cc;
Database changed

mysql> create table cc_t1(data int);
Query OK, 0 rows affected (0.01 sec)

mysql> use tt;
Database changed
mysql> create table tt_t1(data int);
Query OK, 0 rows affected (0.01 sec)
```

步骤 06 登录从数据库，查询数据库 `cc` 库和 `tt` 库相应的数据是否更新。

```
[root@localhost ~]# mysql -u root -P 3307 -S /tmp/mysql2.sock
```

```
Enter password:
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 11
```

```
Server version: 5.6.10-log Source distribution
```

```
Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| cc                 |
| mysql              |
| test               |
| tt                 |
+-----+
```

```
5 rows in set (0.00 sec)
```

```
mysql> use cc;
```

```
Database changed
```

```
mysql> show tables;
```

```
+-----+
| Tables_in_cc      |
+-----+
| ccl                |
| cc_t1              |
+-----+
```

```
2 rows in set (0.00 sec)
```

```
mysql> use tt;
```

```
Database changed
```

```
mysql> show tables;
```

```
Empty set (0.00 sec)
```

10.4 查看 Slave 的复制进度

很多情况下,用户都想知道从服务器复制的进度,从而判断从服务器上复制数据的完整性,同时判断是否需要手工来做主从的同步工作。事实上,用户可以通过 SHOW PROCESSLIST 列表中的 Slave_SQL_Running 线程的 Time 值得到,它记录了从服务器当前执行的 SQL 时间戳与系统时间之间的差距,下面通过例子测试一下这个时间的准确性。

步骤 01 在主服务器上插入一个包含当前时间戳的记录,命令执行如下。

```
mysql> alter table rep_t3 add column createtime datetime;
Query OK, 0 rows affected (0.10 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> insert into rep_t3 values(1,now());
Query OK, 1 row affected (0.00 sec)

mysql> select * from rep_t3;
+-----+-----+
| data | createtime          |
+-----+-----+
| 1 | 2012-07-25 15:10:51 |
+-----+-----+
1 row in set (0.01 sec)
```

步骤 02 让从服务器的 I/O 线程停止下来,使得从数据库服务器暂时不写中继日志,停止时执行的 SQL 就是最后执行的 SQL,命令执行如下。

```
mysql> stop slave;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from rep_t3;
+-----+-----+
| data | createtime          |
+-----+-----+
| 1 | 2012-07-25 15:10:51 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select now();
+-----+-----+
| now()          |
+-----+-----+
| 2012-07-25 15:13:42 |
+-----+-----+
```



```
1 row in set (0.00 sec)
```

步骤 03 从数据库服务器上执行 `show processlist` 查看 SQL 线程的时间, 这个时间说明了主服务器最后执行的更新操作大概是主服务器 46 秒前的更新操作, 命令执行如下。

```
mysql> stop slave io_thread;
Query OK, 0 rows affected (0.00 sec)

mysql> show processlist \G;
***** 1. row *****
      Id: 6
     User: root
    Host: localhost
       db: test
 Command: Query
      Time: 0
     State: NULL
      Info: show processlist
***** 2. row *****
      Id: 8
     User: system user
    Host:
       db: NULL
 Command: Connect
      Time: 46
     State: Has read all relay log; waiting for the slave I/O thread to update it
      Info: NULL
2 rows in set (0.00 sec)
```

10.5 日常管理和维护

数据复制环境配置完成后, 数据库管理员需要进行日常的监控和管理维护工作, 以便能够及时发现问题和解决问题, 以此来保证主从数据库能够正常地工作。有时候因为主服务器的更新过于频繁, 造成了从服务器更新速度较慢, 当然问题是多种多样, 有可能是网络搭建的结构不好或者硬件的性能较差, 从而使得主从服务器之间的差距越来越大, 最终对某些应用产生了影响, 在这种情况下, 用户需要定期进行主从服务器的数据同步操作。

10.5.1 了解服务器的状态

一般使用 `show slave status` 命令来检查从服务器, 如下例所示。

```
mysql> show slave status\G;
```

```

***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 127.0.0.1
Master_User: repl
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000013
Read_Master_Log_Pos: 98
Relay_Log_File: mysql2-relay-bin.000035
Relay_Log_Pos: 235
Relay_Master_Log_File: mysql-bin.000013
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table: test.rep_t1,test.rep_t1
Replicate_Ignore_Table: test.rep_t2,test.rep_t2
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 98
Relay_Log_Space: 235
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
1 row in set (0.00 sec)

```

在查看从服务器信息中，首先要查看“Slave_IO_Running”和“Slave_SQL_Running”这两个进程状态是否是“yes”，Slave_IO_Running 表明，是否此进程能够由从服务器到主服务器上正确地读取 BINLOG 日志，并写入到从服务器的中继日志中。Slave_SQL_Running 则表明，能否读取并执行中继日志中的 BINGOG 信息。

10.5.2 服务器复制出错的原因

在某些情况下，会出现从服务器更新失败，此时，首先需要确定是否是主从服务器的表不

同造成的。如果是表结构不同导致的，则修改从服务器上的表与主服务器上的表一致，然后重新执行 START SLAVE 命令。服务器复制出错的常见问题如下。

问题一：出现 “log event entry exceeded max_allowed_pack” 错误。

如果在应用中出现使用大的 BLOB 列或者长字符串，那么在从服务器上复制时，可能会出现 “log event entry exceeded max_allowed_pack” 的错误，这是因为含有大文本的记录无法通过网络进行传输而导致错误，解决方法是在主从服务器上添加 max_allowed_packet 参数，该参数默认设置是 1MB，如下。

```
mysql> SHOW VARIABLES LIKE 'MAX_ALLOWED_PACKET';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| max_allowed_packet | 1048576 |
+-----+-----+
1 row in set (0.00 sec)

mysql> SET @@global.max_allowed_packet=16777216;
Query OK, 0 rows affected (0.00 sec)
```

同时在 my.cnf 里，设置 max_allowed_packet=16MB，数据库重新启动之后该参数将有效。

问题二：多主复制时的自增长变量冲突问题。

大多数情况下使用一台主服务器对一台或者多台从服务器，但是在某些情况下，可能会存在多个服务器配置为复制主服务器，使用 auto_increment 时应采取特殊步骤以防止键值冲突，否则插入行时多个主服务器会试图使用相同的 auto_increment 值。

服务器变量 auto_increment_increment 和 auto_increment_offset 可以协调多主服务器复制和 auto_increment 列。

在多主服务器复制到从服务器的过程中，迟早会发生主键冲突，为了解决这种情况，将不同的主服务器的这两个参数重新设置，可以将 A 数据库服务器设置为 auto_increment_increment=1, auto_increment_offset=1, 此时 B 数据库服务器设置为 auto_increment_increment=1, auto_increment_offset=0。

下面的例子演示修改这两个参数后的效果，具体操作步骤如下。

步骤 01 创建表 auto_t，系统默认的 auto_increment_increment 和 auto_increment_offset 参数都是 1，默认增加幅度为 1，命令执行如下。

```
mysql> create table auto_t( data int primary key auto_increment )engine=myisam
default charset=gbk;
Query OK, 0 rows affected (0.05 sec)

mysql> show variables like 'auto_inc%';
```



```

+-----+-----+
| Variable_name      | Value |
+-----+-----+
| auto_increment_increment | 1     |
| auto_increment_offset   | 1     |
+-----+-----+
2 rows in set (0.00 sec)

```

```

mysql> insert into auto_t values(null),(null),(null);
Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

```

```

mysql> select * from auto_t;
+-----+
| data |
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.00 sec)

```

步骤 02 重新设置参数 `auto_increment_increment` 的值为 10，然后插入数据，命令执行如下。

```

mysql> set @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

```

```

mysql> show variables like 'auto_inc%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| auto_increment_increment | 10    |
| auto_increment_offset   | 1     |
+-----+-----+
2 rows in set (0.00 sec)

```

```

mysql> insert into auto_t values(null),(null),(null);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

```

```

mysql> select * from auto_t;
+-----+
| data |
+-----+
| 1 |

```

```

| 2 |
| 3 |
| 11 |
| 21 |
| 31 |
+-----+
6 rows in set (0.00 sec)

```

从测试效果看，每次递增值是 10，下面看参数 `auto_increment_offset` 的用法。

步骤 03 重新设置参数 `auto_increment_offset` 的值为 5，再插入数据，命令执行如下。

```

mysql> set @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into auto_t values(null),(null),(null);
Query OK, 3 rows affected (0.00 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> select * from auto_t;
+-----+
| data |
+-----+
| 1 |
| 2 |
| 3 |
| 11 |
| 21 |
| 31 |
| 35 |
| 45 |
| 55 |
+-----+
9 rows in set (0.00 sec)

```

从插入的记录可以看出，`auto_increment_increment` 参数是每次增加的量，而参数 `auto_increment_offset` 参数设置的是每次增加后的偏移量，也就是每次按照 10 累加后，还需要增加 5 个偏移量。

10.6 切换主从服务器

在实际工作环境中，有时候遇到这样的问题，有一个这样的工作环境，一个主数据库服务器 A，两个从数据库服务器 B、C 同时指向主数据库服务器，当主数据库服务器 A 发生故障

时，需要将其中的一个从数据库 B 服务器切换成主数据库，同时修改数据库 C 服务器的配置，使其指向新的主数据库 B。

下面介绍一下切换主从服务器的具体操作步骤。

步骤 01 首先要确保所有的从数据库都已经执行了 relay log 中的全部更新，看从数据库的状态是否是 Has read all relay log，是否更新都已经执行完成。

```
mysql> stop slave IO_THREAD;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW PROCESSLIST \G;
***** 1. row *****
      Id: 2
     User: system user
      Host:
       db: NULL
Command: Connect
      Time: 39
   State: Has read all relay log; waiting for the slave I/O thread to update it
     Info: NULL
***** 2. row *****
      Id: 3
     User: root
      Host: localhost
       db: NULL
Command: Query
      Time: 0
   State: NULL
     Info: SHOW PROCESSLIST
2 rows in set (0.00 sec)
```

步骤 02 在从数据库 B 上停止 slave 服务，然后执行 reset master 重置成主数据库。

```
mysql> stop slave;
Query OK, 0 rows affected (0.00 sec)

mysql> reset master;
ERROR 1186 (HY000): Binlog closed, cannot RESET MASTER
```

此时发现报错 Binlog 没有设置，不能够执行 reset master 命令，下面关闭数据库服务，然后修改/etc/my.cnf，在[mysql2]后面的配置选项添加 log-bin 选项，修改如下所示。

```
[mysql2]
...
log-bin = /usr/local/var/mysql2/mysql-bin
```

配置完成后，重启数据库服务，登录数据库 B，然后执行如下命令开启主数据库功能。


```
mysql> stop slave;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> reset master;
Query OK, 0 rows affected (0.04 sec)
```

此时从数据库 B 已经成功切换成为主数据库，下面接着设置从数据库。

步骤 03 在从数据库 B 上添加具有 replication 权限的用户 repl，查询主数据库状态，命令执行如下。

```
mysql> grant replication slave on *.* to 'repl'@'localhost' identified by '123';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> show master status;
```

| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
|------------------|----------|--------------|------------------|
| mysql-bin.000001 | 229 | | |

1 row in set (0.00 sec)

步骤 04 在从数据库 C 上配置复制的参数，具体配置如下。

```
mysql> change master to
-> master_host='127.0.0.1',
-> master_user='repl',
-> master_password='123',
-> master_port=3307,
-> master_log_file='mysql-bin.000002',
-> master_log_pos=98;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> start slave;
Query OK, 0 rows affected (0.01 sec)
```

步骤 05 在从数据库 C 上执行 show slave status 命令查看从数据库服务是否成功启动。

```
mysql> show slave status \G;
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 127.0.0.1
Master_User: repl
Master_Port: 3307
Connect_Retry: 60
Master_Log_File: mysql-bin.000002
Read_Master_Log_Pos: 98
Relay_Log_File: mysql3-relay-bin.000002
```

```

Relay_Log_Pos: 235
Relay_Master_Log_File: mysql-bin.000002
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Errno: 0
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 98
Relay_Log_Space: 235
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
1 row in set (0.00 sec)

```

步骤 06 在主数据库 B 和从数据库 C 上面测试数据库是否成功设置复制功能，首先查看主数据库 B 中 test 库中的表的情况，命令执行如下。

```

mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| rep_t1          |
| rep_t2          |
+-----+
2 rows in set (0.00 sec)

```

查询从数据库 C 中 test 库里表的情况，命令执行如下。

```

mysql> use test;
Database changed
mysql> show tables;
Empty set (0.01 sec)

```

步骤 07 在主数据库 B 中增加表 rep_t3，命令执行如下。

```
mysql> create table rep_t3(data int);  
Query OK, 0 rows affected (0.01 sec)
```

步骤 08 在从数据库 C 中查询，看表是否成功复制到从数据库，命令执行如下。

```
mysql> show tables;  
+-----+  
| Tables_in_test |  
+-----+  
| rep_t3          |  
+-----+  
1 row in set (0.00 sec)
```

至此，主从数据库成功地发生切换。最后，如果主数据库 A 可以修复，可以考虑采用以上方法将 A 数据库配置成为 B 数据库的从数据库。

10.7 小结

本章节主要讲述了 MySQL Replication 的技术，包括在 Windows 环境下如何实现主从复制操作、在 Linux 环境下如何完成 MySQL 的复制操作、如何查看 Slave 的复制进度、服务器如何管理和维护和切换主从服务器的方法。通过本章节的学习，读者可以轻松地实现主从服务器之间的复制同步工作。

第 11 章

◀ MySQL Cluster 实战 ▶

MySQL Cluster 技术在分布式系统中为 MySQL 提供了冗余特性,增强了安全性,可以大大提高系统的可靠性和数据的有效性。MySQL 集群需要一组计算机,每台计算机可以理解为一个节点,这些节点的功能各不相同。MySQL Cluster 按照功能来分,可以分为三种节点:管理节点、数据节点和 SQL 节点。集群中的某台计算机可以是某一个节点,也可以是两种或者三种节点的集合,这些节点组合在一起,为应用提供具有高可靠性、高性能的 Cluster 数据管理。

11.1 MySQL Cluster 概述

目前企业数据量越来越大,所以对 MySQL 的要求进一步提高。以前的大部分高可用方案通常存在一定的缺陷,例如 MySQL Replication 方案,Master 是否存活检测需要一定的时间,如果需要主从切换也需要一定的时间,因此高可用很大程度上依赖于监控软件和自动化工具。随着 MySQL Cluster 的不断发展,终于在性能和高可用上得到了很大的提高。

11.1.1 MySQL Cluster 基本概念

MySQL Cluster 简单地讲是一种 MySQL 集群的技术,是由一组计算机构成,每台计算机可以存放一个或者多个节点,其中包括 MySQL 服务器,DNB Cluster 的数据节点,管理其他节点,以及专门的数据访问程序,这些节点组合在一起,就可以为应用提供了高性能、高可用性和可缩放性的 Cluster 数据管理。

MySQL Cluster 的访问过程大致是这样的,应用通常使用一定的负载均衡算法将对数据访问分散到不同的 SQL 节点,SQL 节点对数据节点进行数据访问并从数据节点返回数据结果,管理节点仅仅只是对 SQL 节点和数据节点进行配置管理,MySQL Cluster 的系统架构如图 11-1 所示。

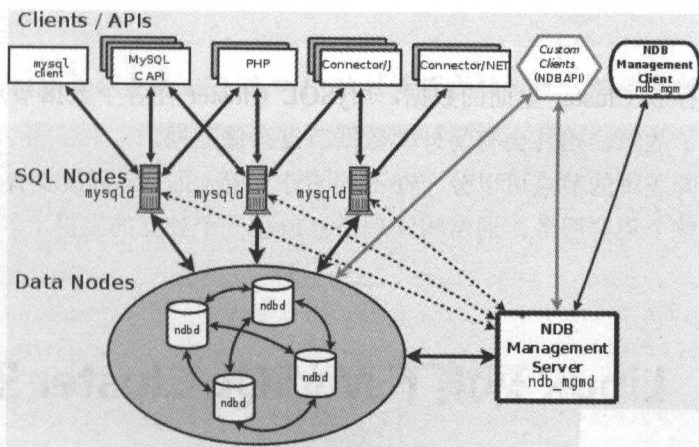


图 11-1 MySQL Cluster 架构

11.1.2 理解 MySQL Cluster 节点

MySQL Cluster 按照节点类型可以分为 3 种类型的节点，分别是管理节点、SQL 节点、数据节点，所有的这些节点构成了一个完整的 MySQL 集群体系。事实上，数据保存在 NDB 存储服务器的存储引擎中，表结构则保存在 MySQL 服务器中，应用程序通过 MySQL 服务器访问数据，而集群管理服务器则通过管理工具 `ndb_mgmd` 来管理 NDB 存储服务器。

下面先了解下 MySQL Cluster 3 种不同类型的节点。

1. 管理节点

管理节点主要用来对其他节点进行管理。通常通过配置 `config.ini` 文件来配置集群中有多少需要维护的副本、配置每个数据节点上为数据和索引分配多少内存、IP 地址，以及在每个数据节点上保存数据的磁盘路径。

管理节点通常管理 Cluster 配置文件和 Cluster 日志。Cluster 中的每个节点从管理服务器检索配置信息，并请求确定管理服务器所在位置的方式。如果节点内出现新的事件时，节点将这类事件的信息传输到管理服务器，将这类信息写入到 Cluster 日志中。

一般在 MySQL Cluster 体系中至少需要一个管理节点，另外值得注意的是，因为数据节点和 SQL 节点在启动之前需要读取 Cluster 的配置信息，所以通常管理节点是最先启动的。

2. SQL 节点

SQL 节点简单地讲就是 `mysqld` 服务器，应用不能直接访问数据节点，只能通过 SQL 节点访问数据节点来返回数据。任何一个 SQL 节点都是连接到所有的存储节点的，所以当任何一个存储节点发生故障的时候，SQL 节点都可以把请求转移到另一个存储节点执行。通常来说，SQL 节点越多越好，SQL 节点越多，分配到每个 SQL 节点的负载就越小，系统的整体性能就越好。

3. 数据节点

数据节点用来存放 Cluster 里面的数据，MySQL Cluster 在各个数据节点之间复制数据，任何一个节点发生了故障，始终会有另外的数据节点存储数据。

通常这 3 种不同逻辑的节点可以分布在不同的计算机上面，集群最少有 3 台计算机，为了保证能够正常维护整个集群服务，通常将管理节点放在一个独立的主机上。

11.2 Linux 环境下 MySQL Cluster 安装和配置

首先准备 3 台普通的 PC 机器，并全部安装好 Fedora 操作系统，本节以 MySQL Cluster 7.2.8 测试环境来介绍下 MySQL Cluster 的配置方法，节点的配置表的信息如表 11-1 所示。

表 11-1 节点配置说明

| 节 点 | 对应的 IP 地址和端口 |
|-------------|--------------------|
| 管理节点（1 个） | 192.168.0.100 |
| SQL 节点（2 个） | 192.168.0.101:3331 |
| | 192.168.0.102:3331 |
| 数据节点（2 个） | 192.168.0.101 |
| | 192.168.0.102 |

下面是安装 MySQL Cluster 7.2.8 之前所需要的准备工作。需要将这 3 台普通 PC 机器的网络 IP 地址配置起来，具体操作步骤如下。

步骤 01 系统默认的网卡端口是 eth0，这里使用的是 eth2，首先编辑 eth2 的配置文件，设置 IP 地址、子网掩码和网关的配置信息。

```
[root@localhost ~]# cd /etc/sysconfig/network-scripts/
[root@localhost network-scripts]# vi ifcfg-eth2
# Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE]
DEVICE=eth2
BOOTPROTO=static
HWADDR=00:0c:29:be:34:3a
ONBOOT=yes
DHCP_HOSTNAME=localhost.localdomain
NM_CONTROLLED=no
TYPE=Ethernet
USERCTL=yes
PEERDNS=yes
IPV6INIT=no
IPADDR=192.168.0.100
NETMASK=255.255.255.0
```



```
GATEWAY=192.168.0.1
```

步骤 02 使用 `ifconfig` 命令查看 `eth2` 端口 IP 地址是否成功设置，然后重新启动网络。

```
[root@localhost network-scripts]# ifconfig eth2
eth2      Link encap:Ethernet  HWaddr 00:0C:29:AD:4D:72
          inet addr:192.168.0.100  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fead:4d72/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:115380 errors:0 dropped:0 overruns:0 frame:0
          TX packets:574 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14511212 (13.8 MiB)  TX bytes:94140 (91.9 KiB)
          Interrupt:18 Base address:0x1080

[root@localhost network-scripts]# service network restart
Shutting down interface eth2: Device eth2 has MAC address 00:0C:29:AD:4D:72,
instead of configured address 00:0C:29:BE:34:3A. Ignoring.
                                                    [FAILED]
Shutting down loopback interface:                  [ OK ]
Bringing up loopback interface:                    [ OK ]
Bringing up interface eth2:                        [ OK ]
```

步骤 03 使用 `chkconfig` 命令设置网卡进入系统时启动。想要每次开机就可以自动获取 IP 地址，此时需要开启服务，使用 `chkconfig` 命令是让网络服务在系统启动级别是 2345 时默认启动。

```
[root@localhost network-scripts]# chkconfig --level 2345 network on
[root@localhost network-scripts]# service network start
Bringing up loopback interface:                    [ OK ]
Bringing up interface eth2:                        [ OK ]
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
RTNETLINK answers: File exists
[root@localhost network-scripts]# clear
```

使用 `service network start` 或者 `service network restart` 命令，提示出错，主要原因是 Fedora 除了有个 `network` 网卡信息外，还有一个 `NetManager` 来管理，可以将 `NetworkManager` 关闭掉。

步骤 04 将 `NetworkManager` 服务关闭掉，然后重启网络服务。

```
[root@localhost network-scripts]# chkconfig --level 0123456 NetworkManager off
[root@localhost network-scripts]# service NetworkManager stop
Stopping NetworkManager daemon: [ OK ]
[root@localhost network-scripts]# service network stop
Shutting down loopback interface: [ OK ]
[root@localhost network-scripts]# service network start
Bringing up loopback interface: [ OK ]
Bringing up interface eth2: [ OK ]
```

下面接着采用相同的方法配置 IP 地址为 192.168.0.101 和 IP 地址为 192.168.0.102 的网络地址，配置完成后我们使用 PING 命令检测下网络是否可以成功连接。

```
[root@localhost ~]# ifconfig
eth2      Link encap:Ethernet  HWaddr 00:0C:29:AD:4D:72
          inet addr:192.168.0.100  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fead:4d72/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:348908 errors:0 dropped:0 overruns:0 frame:0
          TX packets:39045 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:339740810 (324.0 MiB)  TX bytes:2179906 (2.0 MiB)
          Interrupt:18 Base address:0x1080

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1225 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1225 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:73623 (71.8 KiB)  TX bytes:73623 (71.8 KiB)

[root@localhost ~]# ping 192.168.0.101
PING 192.168.0.101 (192.168.0.101) 56(84) bytes of data.
64 bytes from 192.168.0.101: icmp_seq=1 ttl=64 time=4.45 ms
64 bytes from 192.168.0.101: icmp_seq=2 ttl=64 time=1.10 ms
64 bytes from 192.168.0.101: icmp_seq=3 ttl=64 time=2.14 ms
64 bytes from 192.168.0.101: icmp_seq=4 ttl=64 time=1.02 ms
^C
--- 192.168.0.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3975ms
rtt min/avg/max/mdev = 1.022/2.181/4.459/1.387 ms
```

11.2.1 安装 MySQL Cluster 7.2.8 软件

MySQL Cluster 安装之前需要将 MySQL Server 卸载掉，如果 MySQL Server 已经卸载的话，

则直接安装 MySQL Cluster。安装 MySQL Cluster 的具体操作步骤如下。

步骤 01 直接把之前源码安装的 MySQL 程序删除掉。

```
[root@localhost mysql]# cd /usr/local
[root@localhost local]# ls
bin doc etc games include lib libexec man mysql sbin share src
[root@localhost local]# rm -rf mysql
```

步骤 02 登录 <http://dev.mysql.com/downloads/cluster/#downloads> 网址, 下载 MySQL Cluster 7.2.8, 如图 11-2 所示。

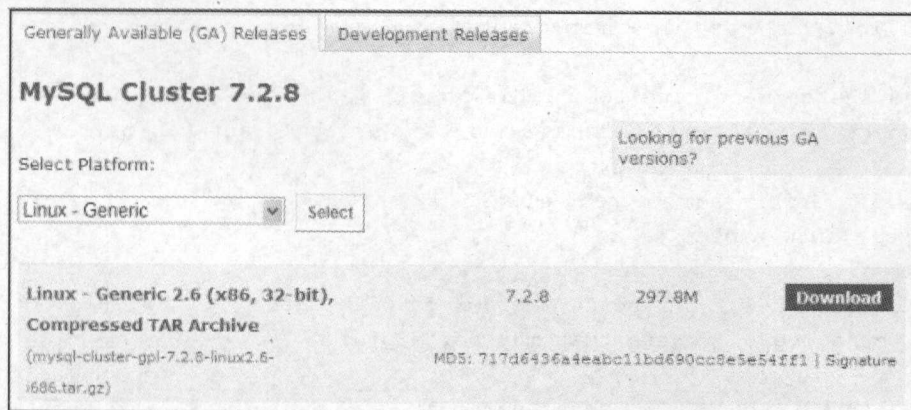


图 11-2 下载 MySQL Cluster

步骤 03 下载好 mysql-cluster-gpl-7.2.8-linux2.6-i686.tar.gz 文件之后, 先对其进行解压缩。

```
[root@localhost ~]# gunzip mysql-cluster-gpl-7.2.8-linux2.6-i686.tar.gz
[root@localhost ~]# tar -xvf mysql-cluster-gpl-7.2.8-linux2.6-i686.tar
```

步骤 04 假设每个节点计算机上都采用 mysql 用户来运行 MySQL Cluster, 首先添加 mysql 组, 然后添加 mysql 用户。

```
[root@localhost ~]# groupadd mysql
[root@localhost ~]# useradd mysql -g mysql
```

步骤 05 开始安装 MySQL Cluster。

```
[root@localhost ~]# mkdir /usr/local/mysql-cluster
[root@localhost ~]# mv mysql-cluster-gpl-7.2.8-linux2.6-i686/*
/usr/local/mysql-cluster/
[root@localhost ~]# cd /usr/local/mysql-cluster
[root@localhost mysql-cluster]# chown -R root .
[root@localhost mysql-cluster]# ls
bin      data  include      lib  mysql-test  scripts  sql-bench
COPYING docs  INSTALL-BINARY  man  README      share    support-files
[root@localhost mysql-cluster]# chown -R mysql ./data
```



```
[root@localhost mysql-cluster]# chown -R mysql .
[root@localhost mysql-cluster]# ./scripts/mysql_install_db --user=mysql
Installing MySQL system tables...
./bin/mysqld: error while loading shared libraries: libaio.so.1: cannot open
shared object file: No such file or directory
```

Installation of system tables failed! Examine the logs in
./data for more information.

You can try to start the mysqld daemon with:

```
shell> ./bin/mysqld --skip-grant &
```

and use the command line tool ./bin/mysql
to connect to the mysql database and look at the grant tables:

```
shell> ./bin/mysql -u root mysql
mysql> show tables
```

Try 'mysqld --help' if you have problems with paths. Using --log
gives you a log in ./data that may be helpful.

Please consult the MySQL manual section
'Problems running mysql_install_db', and the manual section that
describes problems on your OS. Another information source are the
MySQL email archives available at <http://lists.mysql.com/>.

Please check all of the above before mailing us! And remember, if
you do mail us, you MUST use the ./bin/mysqlbug script!

提示错误信息: libaio.so.1: cannot open shared object file: No such file or directory。安装过程中可能缺少的 libaio 安装文件而导致的问题, 接下来, 安装该程序, 操作如下。

```
[root@localhost ~]# ls
anaconda-ks.cfg  install.log.syslog  network.txt
Desktop         libaio-0.3.96-3.i386.rpm  Pictures
Documents       Music                  Public
Download        Mysql5.5              Templates
install.log     mysql-cluster-gpl-7.2.8-linux2.6-i686.tar Videos
[root@localhost ~]# rpm -ivh libaio-0.3.96-3.i386.rpm
warning: libaio-0.3.96-3.i386.rpm: Header V3 DSA signature: NOKEY, key ID
73307de6
Preparing...      ##### [100%]
 1:libaio         ##### [100%]
```

很顺利 libaio-0.3.96-3.i386.rpm 包安装成功, 然后开始初始化 MySQL Server 服务。

```
[root@localhost ~]# cd /usr/local/mysql-cluster/
[root@localhost mysql-cluster]# ./scripts/mysql_install_db --user=mysql
Installing MySQL system tables...
```

OK

Filling help tables...

OK

To start mysqld at boot time you have to copy
support-files/mysql.server to the right place for your system

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !

To do so, start the server, then issue the following commands:

```
./bin/mysqladmin -u root password 'new-password'
```

```
./bin/mysqladmin -u root -h localhost.localdomain password 'new-password'
```

Alternatively you can run:

```
./bin/mysql_secure_installation
```

which will also give you the option of removing the test
databases and anonymous user created by default. This is
strongly recommended for production servers.

See the manual for more instructions.

You can start the MySQL daemon with:

```
cd . ; ./bin/mysqld_safe &
```

You can test the MySQL daemon with mysql-test-run.pl

```
cd ./mysql-test ; perl mysql-test-run.pl
```

Please report any problems with the ./bin/mysqlbug script!

步骤 06 从上面安装信息可以发现 To start mysqld at boot time you have to copy support-files/mysql.server to the right place for your system, 接下来, 首先创建 my.cnf 文件, 并且开始初始化数据库, 接着需要配置 mysql 服务, 然后启动服务。

```
[root@localhost mysql-cluster]# cp ./support-files/my-medium.cnf /etc/my.cnf
cp: overwrite '/etc/my.cnf'? y
[root@localhost mysql-cluster]#
[root@localhost mysql-cluster]# cd /etc/init.d
[root@localhost init.d]# ln -s
/usr/local/mysql-cluster/support-files/mysql.server
/etc/init.d/mysql.server
```

步骤 07 设置 MySQL 服务为自动启动服务。


```
[root@localhost ~]# cd /usr/local
[root@localhost local]# ln -s mysql-cluster mysql
```

接下来编辑/etc/profile 环境配置文件，在该文件最后加上如下的两句配置信息。

```
[root@localhost ~]# vi /etc/profile
PATH=$PATH:/usr/local/mysql-cluster/bin
export PATH
```

步骤 08 使用 chkconfig 增加新的一项服务，系统从其之后服务自动运行。

```
[root@localhost ~]# cd /etc/rc.d/init.d/
[root@localhost init.d]# ls -al |grep mysql
lrwxrwxrwx 1 root root 51 2011-10-27 12:35 mysql.server ->
/usr/local/mysql-cluster/support-files/mysql.server
[root@localhost init.d]# chkconfig --add mysql.server
[root@localhost local]# service mysql.server start
Starting MySQL.....[ OK ]
```

步骤 09 简单地测试下当前的 MySQL 版本是否支持 Cluster。

```
[root@localhost init.d]# mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.5.27-ndb-7.2.8-cluster-gpl-log MySQL Cluster Community
Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show variables like '%ndb%';
```

| Variable_name | Value |
|----------------------|----------|
| have_ndbcluster | DISABLED |
| ndbinfo_database | ndbinfo |
| ndbinfo_max_bytes | 0 |
| ndbinfo_max_rows | 10 |
| ndbinfo_offline | OFF |
| ndbinfo_show_hidden | OFF |
| ndbinfo_table_prefix | ndb\$ |
| ndbinfo_version | 459272 |


```
8 rows in set (0.03 sec)
```

11.2.2 管理节点配置步骤

MySQL Cluster 管理节点的配置是 Cluster 配置中最关键的一步，下面我们通过详细的步骤来描述下如何配置管理节点。

步骤 01 复制/usr/local/mysql-cluster/bin/ndb_mgm,ndb_mgmd 两个文件到/usr/local/bin 目录下。

```
[root@localhost ~]# cd /usr/local/mysql-cluster/bin/
[root@localhost bin]# cp ./ndb_mgm* /usr/local/bin
```

步骤 02 在管理节点服务器 192.168.0.10 的/var/lib/下创建目录 mysql-cluster, 并在该目录下创建配置文件 config.ini。

```
[root@localhost ~]# cd /var/lib
[root@localhost lib]# mkdir mysql-cluster
[root@localhost lib]# cd mysql-cluster/
[root@localhost mysql-cluster]# touch config.ini
```

步骤 03 配置集群的测试环境，config.ini 文件的配置信息如下。

```
[ndbd default]
NoOfReplicas=1           #每个数据节点的镜像数量
DataMemory=200M          #每个数据节点中给数据分配的内存
IndexMemory=20M          #每个数据节点中给索引分配的内存
[ndb_mgmd]               #配置管理节点
NodeId=1
hostname=192.168.0.100
datadir=/var/lib/mysql-cluster/ #管理节点数据(日志)目录
[ndbd]                  #数据节点配置
NodeId=2
hostname=192.168.0.101
datadir=/usr/local/mysql/data/  #数据节点目录
[ndbd]
NodeId=3
hostname=192.168.0.102
datadir=/usr/local/mysql/data/
[mysqld]
hostname=192.168.0.101
[mysqld]
hostname=192.168.0.102
[mysqld]                # Options for mysqld process
```

管理节点通过 config.ini 文件来配置管理节点、SQL 节点和数据节点的信息，通常最关心这 3 类节点的配置，分别定义如下。

- [NDBD DEFAULT]: 表示每个数据节点的默认配置, 在具体的每个节点[NDBD]中不用再写这些选项。
- [NDB_MGMD]: 表示配置管理节点信息。
- [NDBD]: 表示每个数据节点的配置信息, 可以配置多个数据节点信息。
- [MYSQLD]: 表示 SQL 节点的配置信息, 可以有多个, 此节点的个数说明了可以用来连接数据节点的 SQL 节点总数。

11.2.3 配置 SQL 节点和数据节点

SQL 节点和数据节点的配置比较简单, 只需要在 MySQL Server 的配置文件 (my.cnf) 中增加如下内容即可。

```
# The MySQL server
[mysql_cluster]
ndb-connectstring=192.168.0.100      #数据节点定位管理节点的 IP 地址

[mysqld]
Ndbcluster                          #运行 NDB 存储引擎
ndb-connectstring=192.168.0.100      #定位管理节点
port      = 3306
socket     = /tmp/mysql.sock
skip-external-locking
key_buffer_size = 16M
max_allowed_packet = 1M
table_open_cache = 64
sort_buffer_size = 512K
net_buffer_length = 8K
read_buffer_size = 256K
read_rnd_buffer_size = 512K
myisam_sort_buffer_size = 8M
```

在 IP 地址为 192.168.0.101 和 192.168.0.102 的机器上配置数据节点和 SQL 节点的信息, [mysql_cluster]选项配置数据节点的内容, [mysqld]则配置 SQL 节点选项的内容。

11.3 管理 MySQL Cluster

MySQL Cluster 配置完毕后, 接下来详细了解下如何启动、关闭和测试 MySQL Cluster 功能。

11.3.1 Cluster 的启动

MySQL Cluster 需要将集群的各个节点都启动后才能正常运行, 节点的启动的顺序依次是

管理节点、数据节点和 SQL 节点。具体操作步骤如下。

步骤 01 在管理节点(192.168.0.100)上使用 `ndb_mgmd` 命令启动管理节点进程。

```
[root@localhost ~]# cd /var/lib/mysql-cluster/
[root@localhost mysql-cluster]# ndb_mgmd -f ./config.ini
MySQL Cluster Management Server mysql-5.5.27 ndb-7.2.8
```

启动管理节点需要使用 `ndb_mgmd` 命令, `-f` 参数后面是管理节点的配置文件, `ndb_mgmd` 命令实际上就是 MySQL Cluster 管理服务器, 可以通过 `-f` 参数或者 `--config=config_filename` 来指定 MySQL Cluster 的参数文件, 其他选项可以使用 `ndb_mgmd -help` 命令来查看, 下面查看下进程, 判断集群管理进程是否成功启动。

```
[root@localhost ~]# cd /var/lib/mysql-cluster/
[root@localhost mysql-cluster]# ndb_mgmd -f ./config.ini
MySQL Cluster Management Server mysql-5.5.27 ndb-7.2.8

[root@localhost mysql-cluster]# ps -ef |grep ndb
root      9742 15442  0 04:55 pts/1    00:00:00 grep ndb
root      30841      1  0 04:33 ?        00:00:05 ndb_mgmd -f ./config.ini
```

提示

使用 `ps` (进程查看) 命令查看到 `ndb_mgmd` 的进程, 说明 `ndb_mgmd` 进程已经启动。

步骤 02 启动 IP 地址为 192.168.0.101 和 192.168.0.102 的数据节点服务器服务, 如果是第一次启动, 则需要添加 `--initial` 参数, 以便进行 `ndb` 节点的初始化工作。值得注意的是, 在以后的启动过程中, 是不能添加 `--initial` 参数的, 否则 `ndbd` 程序会清除在之前建立的所有用于恢复的数据文件盒日志文件, 如下所示。

```
[root@localhost bin]# ndbd --initial
Unable to connect with connect string: nodeid=0,192.168.0.100:1186
Retrying every 5 seconds. Attempts left: 12 11 10 9 8 7 6 5 4 3 2 1, failed.
2011-10-28 05:46:37 [ndbd] ERROR    -- Could not connect to management server,
error: ''
```

启动数据节点发生问题, 连接不上, 尝试关闭操作系统防火墙。

```
[root@localhost ~]# /etc/init.d/iptables stop
iptables: Flushing firewall rules:                [ OK ]
iptables: Setting chains to policy ACCEPT: filter  [ OK ]
iptables: Unloading modules:                       [ OK ]
[root@localhost ~]# /sbin/chkconfig --level 2345 iptables off
```

提示

`/etc/init.d/iptables stop` 命令可以将系统防火墙暂时关闭掉, 而 `/sbin/chkconfig --level 2345 iptables off` 命令可以让系统启动的时候不启动 `iptables` 服务。

下面重新启动数据节点，效果如下所示。

```
[root@localhost bin]# ndbd --initial
2011-10-23 13:25:02 [ndbd] INFO      -- Angel connected to '192.168.0.100:1186'
2011-10-23 13:25:02 [ndbd] INFO      -- Angel allocated nodeid: 2
```

成功启动数据节点，下面用相同的方法开启 IP 地址为 192.168.0.101。

步骤 03 启动 IP 地址为 192.168.0.101 和 192.168.0.102 的 SQL 节点服务器服务，如下所示。

```
[root@localhost local]# service mysql.server start
Starting MySQL.....[ OK ]
```

步骤 04 节点全部成功启动之后，用 `ndb_mgm` 工具查看集群的状态。

```
[root@localhost ~]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.101 (mysql-5.5.27 ndb-7.2.8, starting, Nodegroup: 0)
id=3 @192.168.0.102 (mysql-5.5.27 ndb-7.2.8, starting, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.100 (mysql-5.5.27 ndb-7.2.8)

[mysqld(API)] 2 node(s)
id=4 @192.168.0.101 (mysql-5.5.27 ndb-7.2.8)
id=5 @192.168.0.102 (mysql-5.5.27 ndb-7.2.8)
```

`ndb_mgm` 工具是 `ndb_mgmd` 的客户管理工具，通过该工具可以很方便地检测 Cluster 的状态、启动备份、关闭 Cluster 等功能。可以通过 `ndb_mgm -help` 命令来查看。

从上面的信息可以很容易地看出已经成功地连接上了管理服务，Connected to Management Server at: localhost:1186，端口开启的是 1186 端口；`[ndb_mgmd(MGM)] 1 node(s)` 则说明了管理节点的信息；`[ndbd(NDB)] 2 node(s)` 则说明了成功连接了 IP 地址分别为 192.168.0.101 和 192.168.0.102 的数据节点；另外 SQL 节点目前处于连接状态的有 2 个。

11.3.2 Cluster 的测试

MySQL Cluster 成功启动之后，下面来测试一下 Cluster 的功能。对于 NDB 存储引擎的数据是会同步的，而其他类型的存储引擎的数据是不会同步到其他数据节点中的。下面针对 MySQL Cluster 中采用 NDB 存储引擎同步数据来进行测试。

1. 测试 NDB 数据引擎

步骤 01 在其中一个 IP 地址为 192.168.0.101 的 SQL 节点的 test 库中创建存储引擎为 NDB 的表 t，然后插入两条数据。

```
mysql> use test;
Database changed
mysql> create table t(
  ->   data integer
  -> )engine = ndb;
Query OK, 0 rows affected, 2 warnings (1.56 sec)

mysql> insert into t values(1);
Query OK, 1 row affected (0.48 sec)

mysql> insert into t values(2);
Query OK, 1 row affected (0.07 sec)
```

步骤 02 在其中一个 IP 地址为 192.168.0.102 中查询 test 库中的 t 表，看下两个 SQL 节点的数据是否是一致的。

```
mysql> select * from t;
+-----+
| data |
+-----+
|    1 |
|    2 |
+-----+
2 rows in set (0.04 sec)
```

步骤 03 在 SQL 节点 192.168.0.101 上将测试表 t 的存储引擎改为 MyISAM，再次插入测试输入，执行如下。

```
mysql> alter table t engine=myisam;
Query OK, 2 rows affected (0.88 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> insert into t values(3);
Query OK, 1 row affected (0.02 sec)
```

步骤 04 在 SQL 节点 192.168.0.102 上重新查询表 t，如下。

```
mysql> select * from t;
ERROR 1412 (HY000): Table definition has changed, please retry transaction
```

步骤 05 在 SQL 节点 192.168.0.101 上将测试表 t 的存储引擎改为 NDB，如下。

```
mysql> alter table t engine=ndb;
Query OK, 3 rows affected (0.88 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

步骤 06 在 SQL 节点 192.168.0.102 上再次查询表 t，结果如下。

```
mysql> select * from t;
```

```

+-----+
| data |
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.04 sec)

```

很显然，对于 MySQL Cluster 会将存储引擎为 NDB 的表数据同步，而其他存储引擎不会将数据同步到其他数据节点。

2. SQL 节点故障测试

步骤 01 将 IP 地址为 192.168.0.101 服务器上的 MySQL 服务停止。

```

[root@localhost mysql-cluster]# service mysql.server stop
Shutting down MySQL..... [ OK ]

```

步骤 02 用 ndb_mgm 工具查看集群的状态。

```

[root@localhost ~]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.101 (mysql-5.5.27 ndb-7.2.8, starting, Nodegroup: 0)
id=3 @192.168.0.102 (mysql-5.5.27 ndb-7.2.8, starting, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.100 (mysql-5.5.27 ndb-7.2.8)

[mysqld(API)] 2 node(s)
id=4 (not connected, accepting connect from 192.168.0.100)
id=5 @192.168.0.102 (mysql-5.5.27 ndb-7.2.8)

```

从查询 Cluster 的状态信息可以发现 192.168.0.101 计算机的 SQL 节点已经断开，但是 IP 地址为 192.168.0.102 的 SQL 节点依然处于连接状态。

步骤 03 在 IP 地址为 192.168.0.102 的计算机上查看表 t。

```

mysql> select * from t;
+-----+
| data |
+-----+
| 1 |

```



```
| 2 |
| 3 |
+-----+
3 rows in set (0.04 sec)
```

从结果不难发现，在 MySQL Cluster 体系中如果 SQL 节点发生了故障，并没有引起数据查询的问题。对于应用而言，可以将对故障节点的访问修改成为对没有故障节点的数据的查询访问。

11.3.3 Cluster 的关闭

MySQL Cluster 关闭只需要使用 `ndb_mgm` 命令。执行如下所示。

```
[root@localhost mysql-cluster]# ndb_mgm -e shutdown
Connected to Management Server at: localhost:1186
Node 2: Cluster shutdown initiated
Node 3: Cluster shutdown initiated
Node 3: Node shutdown completed.
Node 2: Node shutdown completed.
2 NDB Cluster node(s) have shutdown.
Disconnecting to allow management server to shutdown.
```

同时也可以使用 `ndb_mgm` 工具进入管理界面后，使用 `shutdown` 命令。执行如下所示。

```
[root@localhost mysql-cluster]# ndb_mgm
ndb_mgm> shutdown
Node 2: Cluster shutdown initiated
Node 3: Cluster shutdown initiated
Node 3: Node shutdown completed.
Node 2: Node shutdown completed.
Disconnecting to allow management server to shutdown.
```

11.4 维护 MySQL Cluster

了解了 MySQL Cluster 的运行方法后，本节主要介绍 MySQL Cluster 的日常维护工作。首先了解下 MySQL Cluster 基本的管理维护，实际上进入 MySQL Cluster 的命令行管理界面可以做大量的维护工作，如下所示。

```
[root@localhost ~]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: localhost:1186
Cluster Configuration
```

```

-----
[ndbd(NDB)] 2 node(s)
id=2 @192.168.0.101 (mysql-5.5.27 ndb-7.2.8, starting, Nodegroup: 0)
id=3 @192.168.0.102 (mysql-5.5.27 ndb-7.2.8, starting, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.100 (mysql-5.5.27 ndb-7.2.8)

[mysqld(API)] 2 node(s)
id=4 (not connected, accepting connect from 192.168.0.100)
id=5 @192.168.0.102 (mysql-5.5.27 ndb-7.2.8)

```

通过 ndb 控制界面下执行 help 命令查看很多其他的维护管理命令:

```

ndb_mgm> help
-----
NDB Cluster -- Management Client -- Help
-----

HELP                                Print help text
HELP COMMAND                        Print detailed help for COMMAND(e.g. SHOW)
SHOW                                Print information about cluster
CREATE NODEGROUP <id>,<id>...       Add a Nodegroup containing nodes
DROP NODEGROUP <NG>                 Drop nodegroup with id NG
START BACKUP [NOWAIT | WAIT STARTED | WAIT COMPLETED]
START BACKUP [<backup id>] [NOWAIT | WAIT STARTED | WAIT COMPLETED]
START BACKUP [<backup id>] [SNAPSHOTSTART | SNAPSHOTEND] [NOWAIT | WAIT STARTED
| WAIT COMPLETED]
                                Start backup (default WAIT
COMPLETED,SNAPSHOTEND)
ABORT BACKUP <backup id>            Abort backup
SHUTDOWN                            Shutdown all processes in cluster
CLUSTERLOG ON [<severity>] ...      Enable Cluster logging
CLUSTERLOG OFF [<severity>] ...     Disable Cluster logging
CLUSTERLOG TOGGLE [<severity>] ...  Toggle severity filter on/off
CLUSTERLOG INFO                     Print cluster log information
<id> START                          Start data node (started with -n)
<id> RESTART [-n] [-i] [-a] [-f]    Restart data or management server node
<id> STOP [-a] [-f]                 Stop data or management server node
ENTER SINGLE USER MODE <id>        Enter single user mode
EXIT SINGLE USER MODE               Exit single user mode
<id> STATUS                         Print status
<id> CLUSTERLOG {<category>=<level>}+ Set log level for cluster log
PURGE STALE SESSIONS                Reset reserved nodeid's in the mgmt server
CONNECT [<connectstring>]           Connect to management server (reconnect
if already connected)
<id> REPORT <report-type>          Display report for <report-type>

```

QUIT

Quit management client

<severity> = ALERT | CRITICAL | ERROR | WARNING | INFO | DEBUG

<category> = STARTUP | SHUTDOWN | STATISTICS | CHECKPOINT | NODERESTART |
CONNECTION | INFO | ERROR | CONGESTION | DEBUG | BACKUP | SCHEMA

<level> = 0 - 15

<id> = ALL | Any database node id

For detailed help on COMMAND, use HELP COMMAND.

也可以通过在 **help** 后面接命令的名称, 获取该命令的相关信息, 如下所示。

ndb_mgm> help status

NDB Cluster -- Management Client -- Help for STATUS command

STATUS Print status

<id> STATUS Displays status information for the data node <id>
 or for All data nodes.

e.g.

ALL STATUS

1 STATUS

When a node is starting, the start phase will be
listed.

Start Phase Meaning

1 Clear the cluster file system(ndb_<id>_fs).
 This stage occurs only when the --initial option
 has been specified.2 This stage sets up Cluster connections, establishes
 inter-node communications and starts Cluster

heartbeats.

3 The arbitrator node is elected.

4 Initializes a number of internal cluster variables.

5 For an initial start or initial node restart,
 the redo log files are created.

6 If this is an initial start, create internal system

tables.

7 Update internal variables.

8 In a system restart, rebuild all indexes.

9 Update internal variables.

10 The node can be connected by APIs and can receive

events.

11

At this point, event delivery is handed over to the node joining the cluster.

(see manual for more information)

通过执行以上的命令获取节点管理的信息,例如通过在管理节点上通过执行 `restart`、`stop`、`shutdown` 等命令来重启某个节点,或者关闭某个节点。此外,可以通过管理节点对整个 Cluster 环境进行备份,以及通过日志命令进行日志的相关管理。

11.4.1 Cluster 的日志的管理

MySQL Cluster 提供了两种日志,集群日志 (clusterlog) 主要用来记录所有 Cluster 节点生成的日志,节点日志 (node log) 记录了数据节点的本地时间。通常采用集群日志,集群日志记录了所有节点的数据,更方便进行管理。集群日志记录在 `config.ini` 同一个目录下面,测试环境是在 `/var/lib/mysql-cluster/` 目录下面,文件格式为 `ndb_<nodeid>_cluster.log`,详细信息如下所示。

```
[root@localhost mysql-cluster]# cd /var/lib/mysql-cluster/
[root@localhost mysql-cluster]# ls
config.ini ndb_1_cluster.log ndb_1_out.log ndb_1.pid
[root@localhost mysql-cluster]# cat ndb_1_cluster.log
... ..
2011-10-28 12:23:33 [MgmtSrvr] INFO      -- Nodeid 2 allocated for NDB at
192.168.0.101
2011-10-28 12:23:34 [MgmtSrvr] INFO      -- Node 1: Node 2 Connected
2011-10-28 14:03:01 [MgmtSrvr] INFO      -- Shutting down server...
2011-10-28 14:03:34 [MgmtSrvr] INFO      -- Shutdown complete
2011-10-30 04:18:22 [MgmtSrvr] INFO      -- Loaded config from
'/usr/local/mysql/mysql-cluster/ndb_1_config.bin.1'
2011-10-30 04:18:22 [MgmtSrvr] INFO      -- Id: 1, Command port: *:1186
2011-10-30 04:18:23 [MgmtSrvr] INFO      -- Node 1: Node 1 Connected
2011-10-30 04:18:23 [MgmtSrvr] INFO      -- MySQL Cluster Management Server
mysql-5.5.27 ndb-7.2.8 started
2011-10-30 04:18:23 [MgmtSrvr] INFO      -- Node 1 connected
```

通常可以使用 `ndb_mgm` 客户端管理日志信息,如下所示。

(1) 在终端执行 `ndb_mgm` 命令,输入 `clusterlog info` 命令查询当前日志状态。

```
[root@localhost ~]# cd /var/lib/mysql-cluster/
[root@localhost mysql-cluster]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> clusterlog info
Connected to Management Server at: localhost:1186
Severities enabled: INFO WARNING ERROR CRITICAL ALERT
```

(2) 执行 `clusterlog off` 命令关闭日志。

```
ndb_mgm> clusterlog off
Cluster logging is disabled
ndb_mgm> clusterlog info
Cluster logging is disabled.
```

(3) 执行 clusterlog on 命令开启日志。

```
ndb_mgm> clusterlog on
Cluster logging is enabled.
ndb_mgm> clusterlog info
Severities enabled: INFO WARNING ERROR CRITICAL ALERT
```

11.4.2 Cluster 的联机备份

使用 mysqldump 工具对 MySQL 数据库进行逻辑备份。采用这种备份方法同样地适用于 MySQL Cluster 数据备份,在对 MySQL Cluster 接点进行数据备份的时候,可以选取任意一个节点进行备份。

通常使用 MySQL Cluster 备份指的是在给定时间对数据库的快照,备份包含 3 个部分。

- Metadata (元数据): 所有数据库表的名称和定义。
- Table records (表记录): 执行备份时实际保存在数据库表中的数据。
- Transaction log (事务日志): 指明如何以及何时将数据保存在数据库中的连续记录。

使用管理服务器进行 Cluster 物理备份,首先需要启动管理服务器(ndb_mgm),并执行“start backup”命令启动备份,具体执行如下所示。

```
ndb_mgm> start backup;
Waiting for completed, this may take several minutes
Node 3: Backup 1 started from node 1
Node 3: Backup 1 started from node 1 completed
StartGCP: 2385 StopGCP: 2388
#Records: 2054 #LogRecords: 0
Data: 50696 bytes Log: 0 bytes
```

如果在备份的过程中想终止备份可以使用如下的命令。

```
[root@localhost mysql-cluster]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> abort backup node_id
```

对于大量数据的备份,MySQL Cluster 提供了几个参数供调整,这些参数需要在 config.ini 文件的[nbdb]或者[nbdb default]组中,对各参数的解释如下所示。

- BackupDataBufferSize: 将数据写入磁盘之前用于对数据进行缓冲处理的内存量。
- BackupLogBufferSize: 将日志记录写入磁盘之前用于对其进行缓冲处理的内存量。
- BackupMemory: 在数据库节点中为备份分配的总内存。它应是分配给备份数据缓冲的

内存和分配给备份日志缓冲的内存之和。

- BackupWriteSize: 写入磁盘的块大小。它适用于备份数据缓冲和备份日志缓冲。
- BackupMaxWriteSize: 最大写入磁盘的块大小。

上面的备份例子在两个数据节点下都可以看到备份的数据，在 IP 地址为 192.168.0.101 的数据节点下，可以看到以下的数据文件。

```
[root@localhost ~]# cd /usr/local/mysql/data/BACKUP/BACKUP-1
[root@localhost BACKUP-1]# ls
BACKUP-1.2.LOG      BACKUP-1.2.LOG      BACKUP-1-0.2.Data
```

在 IP 地址为 192.168.0.102 的数据节点下，可以看到以下的数据文件。

```
[root@localhost ~]# cd /usr/local/mysql/data/BACKUP/BACKUP-1
[root@localhost BACKUP-1]# ls
BACKUP-1.3.LOG      BACKUP-1.3.LOG      BACKUP-1-0.3.Data
```

11.4.3 Cluster 的数据恢复

使用 START BACKUP 进行 MySQL Cluster 备份，使用 ndb_restore 工具进行数据恢复，下面使用命令 ndb_restore 进行恢复。

(1) 备份之前，首先创建测试表 t，然后添加若干条记录，如下所示。

```
mysql> select count(*) from t;
+-----+
| count(*) |
+-----+
|      8192 |
+-----+
1 row in set (0.03 sec)
```

(2) 在管理节点上执行 start backup 进行数据备份。

```
ndb_mgm> start backup;
Waiting for completed, this may take several minutes
Node 3: Backup 2 started from node 1
Node 3: Backup 2 started from node 1 completed
StartGCP: 3059 StopGCP: 3062
#Records: 10249 #LogRecords: 0
Data: 280592 bytes Log: 0 bytes
```

(3) 在 IP 地址为 192.168.0.101 的数据节点上执行如下数据恢复的命令，如下所示。

```
[root@localhost mysql-cluster]# ndb_restore -b 3 -n 2 -c
host=192.168.0.100:1186 -m -r /usr/local/mysql/data/BACKUP/BACKUP-3
```

(4) 在 IP 地址为 192.168.0.102 的数据节点上执行如下数据恢复的命令，如下所示。


```
[root@localhost mysql-cluster]# ndb_restore -b 3 -n 3 -c
host=192.168.0.100:1186 -m -r /usr/local/mysql/data/BACKUP/BACKUP-3
```

其中，命令行的各参数含义如下表 11-2 所示。

表 11-2 相关参数及其说明

| 参 数 | 说 明 |
|-----|------------------------|
| -b | 备份 id |
| -n | 节点 id |
| -m | 恢复表定义 |
| -r | 恢复路径 |
| -c | Cluster 管理器连接 IP 地址和端口 |

11.5 Windows 操作系统中配置 Cluster

MySQL Cluster 目前支持 Linux、Mac OS X、Solaris 和 Windows 操作系统，本章节采用 Windows 操作系统下的 MySQL Cluster 版本 MySQL Cluster 7.2.8 为例说明 MySQL Cluster 的配置和启动。具体的节点配置如表 11-3 所示。

表 11-3 节点配置说明

| 节 点 | 对应的 IP 地址 |
|--------------|---------------|
| 管理节点 (1 个) | 192.168.0.208 |
| SQL 节点 (2 个) | 192.168.0.102 |
| | 192.168.0.206 |
| 数据节点 (2 个) | 192.168.0.102 |
| | 192.168.0.206 |

可以看到 IP 地址为 192.168.0.206 的机器同时作为 SQL 节点和数据节点，IP 地址为 192.168.0.102 的机器也是这样，如果 IP 地址不一样，则在下面出现的配置文件中的 IP 地址也应该保持一致。

1. MySQL Cluster 的下载

配置集群需要使 MySQL Cluster 满足集群要求，MySQL Cluster 支持 Linux、Mac OS X、Solaris 和 Windows 操作系统。

MySQL Cluster 的下载地址是 <http://www.mysql.com/downloads/cluster>。如果操作系统是 32 位的，就选择 Windows (x86, 32-bit), ZIP Archive 下载；如果是 64 位的，就选择 Windows (x86, 64-bit), ZIP Archive 下载。解压下载的安装包，为了方便，把解压后文件夹的名字改为 mysql。

2. 管理节点的安装配置

管理节点安装在 C 盘下，在 IP 地址为 192.168.0.208 的机器上创建 c:/mysql/bin、c:/mysql/mysql-cluster 和 c:/mysql/bin/cluster-logs 目录，然后将安装包解压后的 mysql/bin 目录中的 ndb_mgmd.exe 和 ndb_mgm.exe 复制到 IP 地址为 192.168.0.208 的 c:/mysql/bin 目录下。然后在 192.168.0.208 的 c:/mysql/bin 下生成 my.ini 和 config.ini 两个文件。

my.ini 的配置信息如下：

```
[mysql_cluster]
#Option for management node process
config-file=C:/mysql/bin/config.ini
```

config.ini 的配置信息如下：

```
[ndbd default]
NoOfReplicas=2                #每个数据节点的镜像数量
DataDir=D:/Program Files/mysqlcluster/datanode/mysql/bin/cluster-data
DataMemory=80M                #每个数据节点中给数据分配的内存
IndexMemory=18M               #每个数据节点中给索引分配的内存

[ndb_mgmd]
HostName=192.168.0.208        #管理节点的 IP 地址
DataDir=C:/mysql/bin/cluster-logs #管理节点日志文件的目录

[ndbd]
HostName=192.168.0.102        #192.168.0.102的主机作为数据节点

[ndbd]
HostName=192.168.0.206        #192.168.0.206的主机作为数据节点

[mysqld]
HostName=192.168.0.102        #192.168.0.102的主机作为 SQL 节点
[mysqld]
HostName=192.168.0.206        #192.168.0.206的主机作为 SQL 节点
```

3. 数据节点的安装配置

将 IP 地址为 192.168.0.102 和 192.168.0.206 的两台机器同时作为数据节点，在 IP 地址为 192.168.0.206 的计算机上创建以下目录结构。

```
D:/Program Files/mysqlcluster/datanode/mysql/bin
D:/Program Files/mysqlcluster/datanode/mysql/cluster-data
D:/Program Files/mysqlcluster/datanode/mysql/bin/cluster-data
```

此时需要在下载的目录解压之后的文件 mysql/bin 中将 ndbd.exe 复制到 192.168.0.206 的

D:/Program Files/mysqlcluster/datanode/mysql/bin 目录下。接下来需要在该目录（D:/Program Files/mysqlcluster/datanode/mysql/bin）下创建 my.ini 文件，my.ini 的配置信息如下：

```
[mysql_cluster]
ndb-connectstring=192.168.0.208           #定位管理节点
```

此时可以直接按照以上步骤在 IP 地址为 192.168.0.102 的计算机上生成相应的目录和文件，由于数据节点的配置都是一样的，所以可以直接 192.168.1.206 的 D:/Program Files/mysqlcluster/datanode 整个文件夹复制到 192.168.1.102 计算机的同样路径下。

4. SQL 节点的安装配置

在 IP 地址为 192.168.0.206 的计算机上创建 D:/Program Files/mysqlcluster/sqlnode 目录，然后将下载解压后的文件夹 mysql 直接整个复制到 D:/Program Files/mysqlcluster/sqlnode 目录下面，然后在 D:/Program Files/mysqlcluster/sqlnode/mysql 下面创建 my.ini 文件，my.ini 的配置信息如下：

```
[mysqld]
ndbcluster                               #运行 NDB 存储引擎
ndb-connectstring=192.168.0.208         #定位管理节点
```

此时我们可以将 IP 地址为 192.168.0.206 的计算机上 D:/Program Files/mysqlcluster/sqlnode 目录复制到 192.168.0.102 相同的目录下面去。

5. 启动管理节点

启动各个节点是有顺序的，先是 Management node，然后是 Data nodes，最后是 SQL nodes。

首先在 IP 地址为 192.168.0.208 的主机上打开命令窗口，切换到 C:\mysql\bin 目录下面，输入：

```
C:\Documents and Settings\Administrator>cd c:\mysql\bin
C:\mysql\bin>ndb_mgmd -f config.ini --configdir=c:\mysql\mysql-cluster
MySQL Cluster Management Server mysql-5.5.27 ndb-7.2.8
```

此时命令窗口不能关闭，否则服务会停止。

6. 启动数据节点

在 IP 地址为 192.168.0.206 的主机中打开一个新的命令行窗口，切换到目录 D:/Program Files/mysqlcluster/datanode/mysql/bin，输入如下命令：

```
D:\Program Files\mysqlcluster\datanode\mysql\bin>ndbd -c 192.168.0.208
2011-10-20 13:25:02 [ndbd] INFO      -- Angel connected to '192.168.0.208:1186'
2011-10-20 13:25:02 [ndbd] INFO      -- Angel allocated nodeid: 3
```

同时，登录在 IP 地址为 192.168.0.102 的主机中打开一个新的命令行窗口，输入如下命令：


```
D:\Program Files\mysqlcluster\datanode\mysql\bin>ndbd -c 192.168.0.102
2011-10-20 13:26:40 [ndbd] INFO      -- Angel connected to '192.168.0.208:1186'
2011-10-20 13:26:40 [ndbd] INFO      -- Angel allocated nodeid: 2
```

下面在 IP 地址为 192.168.0.208 的主机上使用 `ndb_mgm` 命令测试有没有成功连接，如下命令：

```
C:\mysql\bin>ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> all status
Connected to Management Server at: localhost:1186
Node 2: started (mysql-5.5.27 ndb-7.2.8)
Node 3: started (mysql-5.5.27 ndb-7.2.8)
```

此时发现数据节点已经成功启动。

7. 启动 SQL 节点

在 IP 地址为 192.168.0.206 的主机中打开一个新的命令行窗口，切换到目录 `D:/Program Files/mysqlcluster/sqlnode/mysql/bin`，输入如下命令：

```
D:\Program Files\mysqlcluster\sqlnode\mysql\bin>mysqld --console
121020 14:26:31 [Note] Plugin 'FEDERATED' is disabled.
121020 14:26:31 InnoDB: The InnoDB memory heap is disabled
121020 14:26:31 InnoDB: Mutexes and rw_locks use Windows interlocked functions
121020 14:26:31 InnoDB: Compressed tables use zlib 1.2.3
121020 14:26:31 InnoDB: Initializing buffer pool, size = 128.0M
.....
2011-10-20 14:26:40 [NdbApi] INFO  -- Flushing incomplete GCI:s < 1741/3
121020 14:26:40 [Note] NDB Binlog: starting log at epoch 1741/3
121020 14:26:40 [Note] NDB Binlog: ndb tables writable
```

同理，登录 IP 地址为 192.168.0.102 的主机打开一个新的命令行窗口，输入相同的命令启动 SQL 节点。

接下来，可以在管理节点 192.168.0.208 这台计算机上，在命令窗口切换到 `c:/mysql/bin` 目录下输入以下命令来检测数据节点和 SQL 节点是否成功连接，如下所示：

```
C:\mysql\bin>ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: localhost:1186
Cluster Configuration
-----
[ndbd(NDB)]      2 node(s)
id=2      @192.168.0.102  (mysql-5.5.27 ndb-7.2.8, Nodegroup: 0, Master)
id=3      @192.168.0.206  (mysql-5.5.27 ndb-7.2.8, Nodegroup: 0)
```

```
[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.0.208 (mysql-5.5.27 ndb-7.2.8)
```

```
[mysqld(API)] 2 node(s)
id=4 @192.168.0.102 (mysql-5.5.27 ndb-7.2.8)
id=5 @192.168.0.206 (mysql-5.5.27 ndb-7.2.8)
```

8. 下面进行简单的 SQL 节点数据同步测试

首先，在 IP 地址为 192.168.0.102 机器上面登录 MySQL Server，然后创建数据引擎为 ndb 的测试表 t，并添加一条记录，如下所示。

```
D:\Program Files\mysqlcluster\sqlnode\mysql\bin>mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.27-ndb-7.2.8-cluster-gpl MySQL Cluster Community Server
(GPL
)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test;
Database changed
mysql> show tables;
Empty set (0.06 sec)

mysql> create table t(
-> id integer,
-> data varchar(20)
-> )engine=ndb;
Query OK, 0 rows affected (1.34 sec)

mysql> insert into t values(1,'a');
Query OK, 1 row affected (0.05 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

接下来，登录 IP 地址为 192.168.0.206 的计算机，查看节点数据是否同步，如下所示。

```

D:\Program Files\mysqlcluster\sqlnode\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.27-ndb-7.2.8-cluster-gpl MySQL Cluster Community Server
(GPL
)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use test;
Database changed
mysql> show tables;
Empty set (0.06 sec)

mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| t               |
+-----+
1 row in set (0.05 sec)

mysql> select * from t;
+-----+-----+
| id  | data |
+-----+-----+
|  1  | a    |
+-----+-----+
1 row in set (0.34 sec)

mysql>

```

此时 MySQL Cluster 正常运行，数据同步顺利。

11.6 小结

本章节主要讲述了 MySQL Cluster 技术，包括 Linux 环境下如何实现和管理集群、如何维护集群和在 windows 操作系统下配置集群的方法。通过本章节的学习，读者可以轻松地实现集群的管理和维护等操作。

第 12 章

◀ 企业中MySQL的高可用架构 ▶

对于企业应用而言,数据库的持续可用性和可访问性非常重要,尤其是某些互联网企业用户,数据库提供持续可靠的可用性,才会给企业带来良好的效益,为其客户提供优质可靠的服务体验。因此,设计数据库架构初期就需要考虑如何构建一套适合自身应用程序的高可用架构。

MySQL 数据库作为最流行的开源数据库产品,拥有许多成熟的高可用架构方案,其方案的可用性覆盖率为 90%~99.999%,能够适用于对可用性级别的多种不同需求。其主要是利用复制技术,多个不同数据库主机之间进行复制,以保持数据的一致性,并通过一些第三方开源软件来实现负载均衡和统一的访问接口,既减少应用程序开发的复杂性,也降低了企业的运营成本。本章会对 MySQL 两种常用的高可用架构方案进行配置指导。

12.1 MySQL 高可用的简单介绍

在单点访问服务中,客户访问应用系统,应用系统直接访问数据库,这种系统架构中所有存储数据和读写操作都发生在唯一的一台服务器主机上,数据库数据都存储在一台数据库系统中,往往一旦数据库系统发生故障,应用系统没有办法在短时间内恢复正常。

数据库系统往往是所有企业的核心系统,它存储着企业客户资料、生成数据和业务数据,一旦发生意外的停机,在没有及时地恢复上线或者冗余方案的情况下,往往会带来最直接的经济损失。通常使用数据库高可用性的设计,同步读写分离,负载均衡等手段达到减少系统的停机时间,提高系统服务可靠性。下面将具体讲解如何实现数据库的高可用。

12.2 MySQL 主从复制

MySQL 主从复制指单台 MySQL 服务器的数据复制到另一台 MySQL 服务器。这种方案不需要复杂的配置,数据可以从单台 Master 主机复制到任意数量的 Slave 主机。复制使用异步方式,在 MySQL 5.5 中增加了半同步复制,有效地提高了复制的可靠性。

12.2.1 MySQL 主从架构设计

该架构由单台主服务器和多台从服务器构成，主服务器主要接受来自应用程序的写请求，从服务器接受读请求，在从服务器与应用层之间可以搭建负载均衡设备。例如：LVS、Haproxy、F5 等。该架构可将读写请求分离到不同的机器上执行，但需要应用程序实现不同的连接池，将读操作负载均衡到多个从服务器主机上，提高系统整体读操作的处理能力。但只有单台主服务器可以写数据，所以写操作无法负载均衡，也限制了其扩展性，并且当主服务器发生故障时，系统将无法写数据，存在单点故障。通常适用于对可用性要求不高的场景，例如：在线备份，将其中一台从服务器用于备份数据，避免了备份对主服务器产生的影响。

从图 12-1 中可以看出应用程序可以访问多个从服务器主机，压力可以分布在不同的从服务器主机上，在应用层和从服务器中间可以添加 LVS，便可实现负载均衡，同时从服务器可以水平扩展来提高系统整体读性能。

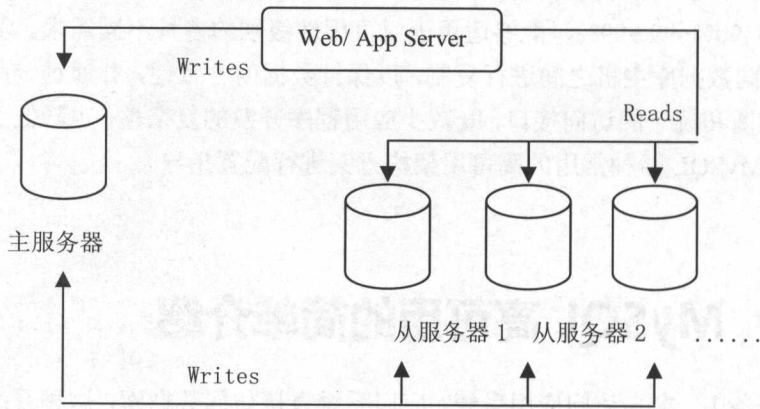


图 12-1 单主多从示意图

12.2.2 配置环境

下面准备了 4 台机器，一台为主服务器、两台从服务器，一台用来做负载均衡。具体的设置如表 12-1 所示。

表 12-1 环境配置表

| 主机名 | IP 地址 | 角色 | 备注 |
|--------|--------------|-----|------|
| 主服务器 | 192.168.1.20 | 主机 | 数据库写 |
| 从服务器 1 | 192.168.1.21 | 从机 | 数据库读 |
| 从服务器 2 | 192.168.1.22 | 从机 | 数据库读 |
| LVS | 192.168.1.23 | LVS | 负载均衡 |

12.2.3 服务器的安装配置

首先在主服务器，从服务器 1，从服务器 2 三台主机上安装 MySQL 数据库。把 MySQL

安装脚本上传到需要安装 MySQL 的服务器上，然后执行脚本操作。脚本有三个参数分别是：软件包、安装目录、配置文件，需要指定这三个参数才能顺利安装，如下所示。

```
#####
#   FileName: mysql_install_local.sh
#####

#!/usr/bin/env bash

fullname=$1
prefix=$2
mycnf=$3
fname=`basename $fullname`
realname=`echo "$fname" | awk -F ".tar" '{print $1}'`
if [ -z "$fullname" ]; then
    echo "Usage: $0 /home/xyz/mysql-xxx.tar.gz /path/to/install_prefix
/path/to/my.cnf"
    exit 1;
fi
echo "Create MySQL user"
tt=`grep mysql /etc/passwd`
if [[ -z $tt ]]; then
    groupadd -f mysql
    useradd -g mysql -d /dev/null -s /sbin/nologin mysql
fi
echo "Clean old MySQL"
rm -rf $prefix/mysql
echo "Unpacking..."
if [[ ! -d $prefix ]]; then
    mkdir -p $prefix
fi
tar xzf $fullname -C $prefix 2>> /tmp/mysqlinstall.log
echo "Setting up symlink mysql->mysql-XYZ..."
ln -s $prefix/$realname $prefix/mysql 2>> /tmp/mysqlinstall.log
echo "export PATH=$PATH:$prefix/mysql/bin" >> /etc/profile
source /etc/profile
test -x $prefix/mysql/bin/mysqld
[ $? != 0 ] && exit 1
test -x $prefix/mysql/bin/mysql
[ $? != 0 ] && exit 1
if [[ -f /etc/my.cnf ]]; then
    mv /etc/my.cnf{,.old}
fi
cp $mycnf /etc/my.cnf
echo "Initial MySQL Database"
```



```

if [[ $PWD != $prefix/mysql ]]; then
    cd $prefix/mysql/
fi
./scripts/mysql_install_db --user=mysql --defaults-file=/etc/my.cnf 2>>
/tmp/mysqlinstall.log |grep -i 'ok'
if [[ $? != 0 ]]; then
    echo "initial mysql dataabse failed see /tmp/mysqlinstall.log "; exit 1
fi
cp -f support-files/mysql.server /etc/init.d/mysql
echo "Installing of MySQL is complete"
echo "You can start MySQL server /etc/init.d/mysql start"

```

MySQL 安装完成后，即可对它们进行配置，具体操作步骤如下：

步骤 01 在主服务器主机上开启 binlog 日志，保持 server-id 的唯一性，这步最好安装 mysql 之前就写在配置文件中。

步骤 02 在主服务器上配置复制所需要的账户，如下所示。

```

mysql> grant replication slave on *.* to repl@% identified by 'repl_password';
Query OK, 0 rows affected (0.05 sec)

```

```

mysql> flush privileges;
Query OK, 0 rows affected (0.01 sec)

```

步骤 03 接着，在从服务器 1 和从服务器 2 上分别配置主服务器信息，如下所示。

```

mysql> change master to master_host='192.168.1.20', master_user='repl',
master_password='repl_password';
Query OK, 0 rows affected (0.05 sec)

```

```

mysql> slave start;
Query OK, 0 rows affected (0.01 sec)

```

```

mysql> show slave status\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 192.168.1.20
      Master_User: repl
      Master_Port: 3306
      Connect_Retry: 3
      Master_Log_File: mysql-bin.001
      Read_Master_Log_Pos: 79
      Relay_Log_File: mysql-relay-bin.001
      Relay_Log_Pos: 548
      Relay_Master_Log_File: mysql-bin.001
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes

```

```

Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
      Last_Errno: 0
      Last_Error:
      Skip_Counter: 0
Exec_Master_Log_Pos: 79
      Relay_Log_Space: 552
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
      Master_SSL_Cert:
      Master_SSL_Cipher:
      Master_SSL_Key:
Seconds_Behind_Master: 8
1 row in set (0.00 sec)

```

下面了解一下其中主要的几个变量的含义。

- **Master_Host**: 被复制的主机地址。
- **Master_User**: 在主服务器上创建的用于复制的账户。
- **Master_Port**: 主服务器上 MySQL 运行的端口。
- **Slave_IO_Running**: 显示 I/O 线程运行状态。
- **Slave_SQL_Running**: 显示 SQL 线程运行状态。
- **Last_Error**: 当复制中断时, 此处会显示错误原因, 并由此来修复复制。
- **Seconds_Behind_Master**: 显示从服务器中 I/O 线程与 SQL 线程之间的时间差。可以反应出主从服务器之间的网络快慢。

到此, MySQL 的主从服务器已经配置好了。

12.2.4 LVS 的安装配置

接下来可以配置 LVS, 用来做负载均衡, 具体操作步骤如下:

步骤 01 在 LVS 主机上安装 LVS 相关软件, 可以通过 Yum 的方式, 如下所示。

```
root@LVS#yum install ipvsadm piranha
```

步骤 02 安装完毕, 开始配置 LVS, 编辑 LVS 的配置文件。

```

root@LVS#vi /etc/sysconfig/ha/lvs.cf
serial_no = 26
primary = 192.168.1.10          #配置虚拟 IP
service = lvs
heartbeat = 1
heartbeat_port = 539
keepalive = 6
deadtime = 18
network = direct
debug_level = NONE
monitor_links = 1
syncdaemon = 0
virtual mysql {
    active = 1
    address = 192.168.1.23 eth0:1    #配置 LVS 的主机物理网口 IP
    vip_nmask = 255.255.255.0
    port = 3306
    use_regex = 0
    load_monitor = none
    scheduler = wrr                #负载方式可选 wrr 加权轮询, 或 wlc 加权最小连接
    protocol = tcp
    timeout = 6
    reentry = 15
    quiesce_server = 0
    server Slave-1 {
        address = 192.168.1.21      #Slave-1的主机 IP
        active = 1
        port = 3306
        weight = 1
    }
    server Slave-2 {
        address = 192.168.1.22      #Slave-2的主机 IP
        active = 1
        port = 3306
        weight = 1
    }
}

```

步骤 03 分别在从服务器 1 和从服务器上安装 `arptables_jf`, 进行 `arp` 设置, 设置完成后启动 `arpable_jf` 服务, 并设置其开机自动启动。

```

root@Slave-1# yum install arptables_jf
root@Slave-1# arptables -A IN -d 192.168.1.10 -j DROP
root@Slave-1# arptables -A OUT -s 192.168.1.10 -j mangle --mangle-ip-s
192.168.1.21
root@Slave-1# echo "ip addr add 192.168.1.10 dev lo" >> /etc/rc.local
root@Slave-1# service arptables_jf start

```



```
root@Slave-1# chkconfig --level 2345 arptables_jf on
```

步骤 04 启动 LVS 服务，并设置其开机自动启动。

```
root@LVS# service pulse start
root@LVS# chkconfig --level 2345 pulse on
```

步骤 05 启动 LVS 后，可以通过 ipvsadm 命令来检查 LVS 状态是否正常。

```
root@LVS# ipvsadm -L -n
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP  192.168.1.10:3306 wrr
  -> 192.168.1.21:3306           Route    1      1      0
  -> 192.168.1.22:3306           Route    1      2      0
```

从结果中可以看到，虚拟 IP 为 192.168.1.10，转发端口是 3306，真实主机的 IP 是 192.168.1.21 和 192.168.1.22。当应用程序访问 192.168.1.10 的 3306 端口时，连接会自动地被转发到 21 和 22 这两台机器上，此时再看 LVS 的状态，ActiveConn 的值就会有变化，说明 LVS 已经正常工作了。

到此为止，大家已经学会了如何配置一个最简单的 MySQL 高可用架构。当然这种架构存在单点故障问题，可以为主服务器准备一台备份机器来解决。接下来的章节就在此架构基础上进一步完善它的可用性。

12.3 MySQL+DRBD+HA

由于一些成长性企业的业务通常发展的很迅速，对数据库的扩展性、可用性和性能的要求会更高，上面所讲的普通方案无法满足这种业务快速增长的场景对可用性、可靠性的需求。越来越大的压力会使主从之间的复制出现较大的延迟，从而影响了数据库数据的一致性，降低了它的可靠性。

有什么办法可以解决这样的问题呢？使用 DRBD 技术执行文件级的复制，主机之间能够保持较高的文件一致性，可靠性可以达到 99.99%，此方案更多地考虑到了扩展性，能很好地适应企业业务的快速增长给数据库带来的读写压力。

12.3.1 什么是 DRBD

DRBD (Distributed Replicated Block Device) 简称分布式复制块设备，是一种基于 Linux 系统的软件组件，它是由内核模块和相关程序而组成，用以构建高可用性的集群。当数据写入到本地的 DRBD 设备上的文件时，数据会同时被发送到网络中的另外一台主机之上。

DRBD 实现原理如图 12-2 所示，是通过网络来镜像整个设备。DRBD 复制接收数据，然后写入到本地磁盘，之后会把数据发送给另一台主机，另一台主机将数据写入到本地磁盘，DRBD 通过对故障进行切换操作，从而实现集群的高可用性。

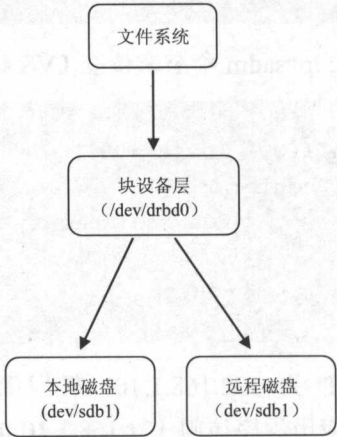


图 12-2 DRBD 工作原理图

12.3.2 MySQL+DRBD+HA 架构设计

该架构主要由 2 台机器组成。分别是 Active 和 Standby。它们之间通过 Heartbeat 进行连接，对外共用一个虚拟 IP 地址，正常情况 Active 对外提供服务，当 Active 出现软硬件单点故障时，Heartbeat 自动将虚拟 IP 指向 Standby，由 Standby 作为新的 Active 对外提供服务。当原 Active 的故障排除后再将其接入系统作为新的 Standby 存在。

Active 和 Standby 之间通过 DRBD 技术实现数据的互为主从的复制，以减少故障切换的时间，详细设计如图 12-3 所示。

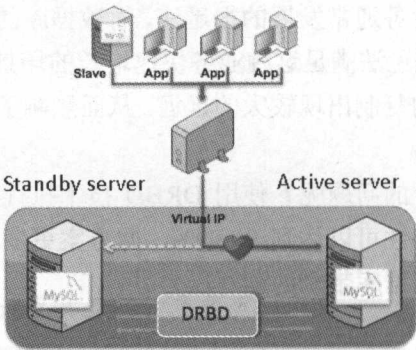


图 12-3 DRBD 方案架构

12.3.3 配置环境

首先我们需要准备两台数据库主机，具体配置信息如表 12-2 所示。

表 12-2 环境配置表

| 主机名 | IP 地址 | 心跳 IP | 备注 |
|-------|--------------|--------------|------------------|
| db-01 | 10.11.196.48 | 192.168.1.11 | VIP:10.11.196.50 |
| db-02 | 10.11.196.49 | 192.168.1.12 | 同上 |

在两台主机上安装好 MySQL 数据库，注意，MySQL 的数据目录需要设定在 DRBD 的块设备的挂载点，以/data 挂载点为例。

12.3.4 安装配置 Heartbeat

安装配置 Heartbeat 的具体操作步骤如下：

步骤 01 分别在两台主机上安装 Heartbeat 心跳软件，可以通过 yum 的方式，如下所示。

```
root@db-01# yum install heartbeat
```

步骤 02 配置 hosts 文件，把主机名与 IP 添入 hosts 文件。

```
root@db-01# cat /etc/hosts
127.0.0.1 localhost.localdomain localhost
::1 localhost6.localdomain6 localhost6
10.11.196.48 db-01
10.11.196.49 db-02
192.168.1.11 db-01
192.168.1.12 db-02
```

步骤 03 配置网卡 IP 和心跳 IP。

```
root@db-01# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
BOOTPROTO=static
IPADDR=10.11.196.48      #另外一台改成10.11.196.49
ONBOOT=yes
root@db-01# cat /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
BOOTPROTO=static
BROADCAST=192.168.1.255
IPADDR=192.168.1.11     #另外一台改成192.168.1.12
NETMASK=255.255.255.0
NETWORK=192.168.1.0
ONBOOT=yes
```

提示

两台数据库主机的系统时间要保持一致，如果内网环境中有 NTP 服务器，可以定期地更新两台数据库服务器的时间；如果没有，那么可以在两台服务器上配置来保持两者的时间同步。

步骤 04 以 db-01 作为 NTP 服务器为例。首先安装 ntpd 软件包，可通过 yum 方式，然后配

置其选项，如下所示。

```
root@db-01# cat /etc/ntp.conf #在 db-01配置 ntpserver
restrict default nomodify notrap noquery
server 127.0.0.1
restrict 192.168.1.0mask 255.255.255.0 nomodify notrap
restrict 127.0.0.1
driftfile /var/lib/ntp/ntp.drift
```

步骤 05 启动 NTP 服务并设置开机自动启动，并在 db-02 上设置定时任务来更新时间。

```
root@db-01# service ntpd start
root@db-01# chkconfig --level 2345 ntpd on

root@db-02# crontab -e
0-59/30 * * * * /usr/sbin/ntpdate 192.168.1.11
```

这些准备工作完成后，就可以开始配置 heartbeat，在 heartbeat 2.x 版本中相关的配置文件有三个，分别是 ha.cf、authkeys、haresources。

步骤 06 首先配置 ha.cf 文件，这个是 heartbeat 核心配置文件。

```
root@db-01# cat /etc/ha.d/ha.cf
debugfile /var/log/ha-debug
logfile /var/log/ha-log
logfacility local0
keepalive 1
deadtime 10
warntime 5
initdead30
udpport 694
bcast eth1 # 此网口必须是心跳口，也可写成 ucast eth1 <ip>对方 ip
auto_failback off #主节点故障恢复后，资源是否自动调回
nodedb-01 db-02 # 节点名称与主机名 (uname -n) 保持一致。
ping10.11.196.254 # 网关
respawn hacluster /usr/lib/heartbeat/ipfail
apiauth ipfail gid=haclient uid=hacluster
deadping 5 # 此时间要小于 deadtime
```

步骤 07 接着配置 authkeys 文件并修改其权限设置，该文件主要是设置通信的验证校验机制。

```
root@db-01# cat/etc/ha.d/authkeys
auth 1
1 crc
root@db-01# chmod 600 /etc/ha.d/authkeys
```

步骤 08 配置 `haresources` 文件，该文件主要是设置挂载的文件系统类型和路径，以及服务名和虚拟 IP。

```
root@db-01# cat/etc/ha.d/haresources
db-01drbddisk Filesystem::/dev/drbd0::/data::ext3 mysql10.11.196.50
主机名 资源名 文件系统: 设备名, 挂载路径, 类型 服务名 虚拟 IP
MailTo::youremail@address.com::DRBDFailure # 此行可选
```

这里需要注意，由于使用的是 DRBD，因此设备名是 `/dev/drbd0`，服务名是 `mysql`，这个名称是需要和 `/etc/init.d/mysql` 脚本名保持一致，否则在发生故障切换时将无法启动 `mysql`。

`heartbeat` 的配置过程在两台数据库主机中都需要设置，记住有些地方需要变动，例如：主机名。

12.3.5 安装配置 DRBD

首先下载源码包，<http://oss.linbit.com/drbd/8.3/drbd-8.3.7.tar.gz>，DRBD 分为内核模块和管理工具两部分，Linux 从 2.6.33 开始就将 DRBD 的驱动模块合并到了内核里，如果你的 Linux 内核大于 2.6.33，就只需要安装 DRBD 管理工具。这里以 8.3.7 版本为例。

步骤 01 分别在两台机器上安装 DRBD，如下所示。

```
root@db-01# wget http://oss.linbit.com/drbd/8.3/drbd-8.3.7.tar.gz
root@db-01# tar xzf drbd-8.3.7.tar.gz
root@db-01# cd drbd-8.3.7
root@db-01# ./configure --with-utils --with-km
root@db-01# make && make install
root@db-01# cp drbd /etc/init.d/
root@db-01# chkconfig --add drbd
```

步骤 02 安装完成后开始配置 DRBD，需要为 DRBD 分配一个块设备，以 `/dev/sdb1` 为例。

```
root@db-01# cat/usr/local/etc/drbd.conf
global {
    usage-count yes;
}
common {
    protocol C;
    syncer {
        rate 100M;
    }
}
resource r0 {
    on db-01 {
        device    /dev/drbd0;
        disk      /dev/sdb1;
        address    192.168.1.11:7789;    #这里填对应主机的心跳地址
```

```

        meta-disk internal;
    }
    on db-02 {
        device    /dev/drbd0;
        disk      /dev/sdb1;
        address    192.168.1.12:7789;
        meta-disk internal;
    }
}

```

步骤 03 配置完成后就可以开始加载内核模块并进行初始化设置（两台机器都需要操作）。

加载 drbd 的内核模块：

```

root@db-01# depmod
root@db-01# modprobe drbd

```

创建 drbd 元数据：

```

root@db-01# drbdadm create-md r0

```

启动 drbd 相关进程：

```

root@db-01# drbdadm attach r0
root@db-01# drbdadm syncer r0
root@db-01# drbdadm connect r0

```

步骤 04 开始创建主节点，并格式化文件系统，注意此步骤只需要在其中一台服务器上操作，这里以 db-01 为例。

```

root@db-01# drbdadm -- --overwrite-data-of-peer primary r0
root@db-01# mkfs.ext3 /dev/drbd0
查看 DRBD 同步状态：
root@db-01# cat /proc/drbd
version: 8.3.7 (api:88/proto:86-91)
GIT-hash: ea9e28dbff98e331a62bcbcc63a6135808fe2917 build by root@db-01,
2012-04-08 16:57:43
0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r----
ns:265140 nr:33016 dw:298156 dr:4089 al:55 bm:15 lo:0 pe:0 ua:0 ap:0 ep:1 wo:d
oos:0

root@db-02# cat /proc/drbd
version: 8.3.7 (api:88/proto:86-91)
GIT-hash: ea9e28dbff98e331a62bcbcc63a6135808fe2917 build by root@db-02,
2012-04-08 16:58:30
0: cs:Connected ro:Secondary/Primary ds:UpToDate/UpToDate C r----
ns:33016 nr:298512 dw:299164 dr:36705 al:8 bm:21 lo:0 pe:0 ua:0 ap:0 ep:1 wo:d
oos:0

```

UpToDate 表示两台服务器的 DRBD 块设备数据已经同步，DRBD 配置成功。注意，DRBD 设备只有在 primary 状态时才可以 mount，如上面的 /proc/drbd 信息所示，当前 db-01 是 primary，

所以可以在 db-01 上挂载/dev/drbd0:

```
root@db-01# mkdir /data
root@db-01# mount /dev/drbd0 /data
```

步骤 05 启动 heartbeat 服务设置开机自动启动。

```
root@db-01# chkconfig --level 234 drbd on
root@db-01# service heartbeat start
```

现在整个配置就完成了, 切记在投入生产环境之前一定要进行一些测试来验证各服务是否运行正常, 模拟故障发生看是否能自动地切换主机, 不会导致服务中断。

设想这样一种场景, 两台 DRBD 主机一主一备, 当两台机器之间的通信出现故障, 备机无法和主机保持同步, 此时 Heartbeat 识别到了连接故障就会将备机切换成主机, 但此时备机的数据并不是和主机完全一致, 这就会导致数据的不一致性。如何来应对这种情况? 启用 Heartbeat 的 dopd 程序, 当出现连接问题两台机器的数据不一致时, 备机的元数据会被设置成 Outdated, 当主机识别到后就不会切换到备机。这样一来就保证了两台机器的数据是一致的, 后面就需要人工去检查问题原因并修复。牺牲了可用性换来数据一致性也是值得的。下面简单介绍如何启用这个功能。

在 Heartbeat 的 ha.cf 配置文件中加入下面内容 (两台主机都需要添加):

```
respawn hacluster /usr/lib/heartbeat/dopd
apiauth dopd gid=haclient uid=hacluster
```

重新加载 Heartbeat 配置, 如下所示。

```
root@db-01# /etc/init.d/heartbeat reload
```

接着, 修改 DRBD 配置文件 drbd.conf 的 common 部分追加加入下面内容:

```
common {
...
    handlers {
        outdate-peer "/usr/lib/heartbeat/drbd-peer-outdater";
    }
}
# 在 resource 部分追加下面内容:
resource r0 {
...
    disk {
        fencing    resource-only;
    }
}
```

重新加载 DRBD 配置文件。

```
root@db-01# drbdadmin adjust all
```

此方案可以作为上一节方案的扩展和增强，将原来的单台主服务器，升级为主备架构，避免了单点故障的发生，在实际的生产环境中往往需要灵活地设计架构才能应对变化多样的业务需求。

12.4 Lvs+Keepalived+MySQL 单点写入主主同步方案

目前在 MySQL 高可用方案中，Lvs+Keepalived 经常被使用到，Lvs 和 Keepalived 为高可用方案提供了负载均衡和故障的转移。通常情况下 MySQL 高可用为了实现数据的一致性，会采用单点写入，本章节采用 keepalived 中的 sorry_server 来实现写入数据库为单点的需求，下面先了解下 Lvs+Keepalived 单点写入主主同步方案的架构图，如图 12-4 所示。

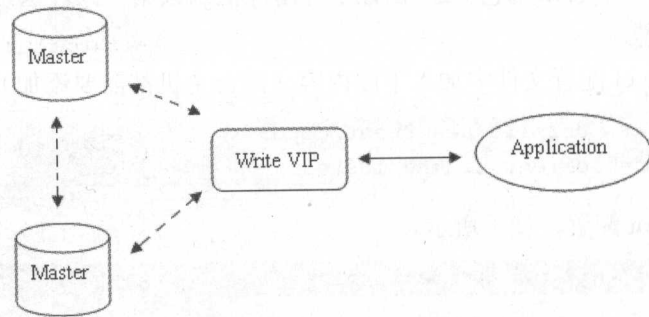


图 12-4 MM+Lvs+Keepalived 单点写入高可用方案

该方案安装配置非常简单，主服务器后面可以添加多个从服务器，并做到负载均衡，事实上该方案中并没有实现数据库的读写分离，只适用于只有两台数据库服务器的情况。

12.4.1 配置环境

下面先准备下 Lvs+Keepalived 高可用方案所需的实战环境，如表 12-3 所示。

表 12-3 Lvs+Keepalived 高可用方案服务器列表

| 服务器 | IP 地址 | VIP | 名称 | MySQL |
|----------|-------------|---------------|-----|--------|
| Master 1 | 192.168.0.2 | 192.168.0.112 | db1 | 5.5.24 |
| Master 2 | 192.168.0.3 | 192.168.0.112 | db2 | 5.5.24 |

首先，先准备两台机器，并安装好 MySQL 服务，然后设置两台服务器主主同步，具体步骤如下。

步骤 01 配置 db1 (192.168.0.2), 如下所示。

```
# Replication Master Server (default)
# binary logging is required for replication
log-bin=mysql-bin

# binary logging format - mixed recommended
binlog_format=mixed

# required unique id between 1 and 2^32 - 1
# defaults to 1 if master-host is not set
# but will not function as a master if omitted
server-id = 1
replicate-do-db = test
slave-skip-errors = all
sync_binlog = 1
log-slave-updates

# set number of masters
auto_increment_increment = 2
auto_increment_offset = 1
```

值得注意的是, log-slave-updates 表示如果一个 Master 挂掉的话, 另一个主机接管。

步骤 02 配置 db2 (192.168.0.3), 如下所示。

```
# Replication Master Server (default)
# binary logging is required for replication
log-bin=mysql-bin

# binary logging format - mixed recommended
binlog_format=mixed

# required unique id between 1 and 2^32 - 1
# defaults to 1 if master-host is not set
# but will not function as a master if omitted
server-id = 2
replicate-do-db = test
slave-skip-errors = all
sync_binlog = 1
log-slave-updates

# set number of masters
auto_increment_increment = 2
auto_increment_offset = 2
```

提示

auto_increment_increment 的值表示整个结构中服务器的总数, 本案例中使用到两台主机服务器, 所以设置为 2。auto_increment_increment 和 auto_increment_offset 用于主主复制防止出现重复的值。

步骤 03 复制 db1 (192.168.0.2) 的数据库到 db2, 保持数据的一致性, 首先将服务器上的表锁定起来, 避免复制过程发生数据的变化, 如下所示。

```
mysql> flush tables with read lock;
Query OK, 0 rows affected (0.00 sec)
```

步骤 04 然后在服务器上查询当前二进制文件的名称以及偏移位置。

```
mysql> show master status;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000022 |      107 |              |                  |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

步骤 05 接着将 db1 数据库的数据进行备份, 然后在 db2 数据上恢复, 这样保持两个数据库一开始数据都是一致的, 如下所示。

```
[root@localhost ~]# cd /usr/local/mysql/bin/
[root@localhost bin]# mysqldump -user=root -p test > /test.sql
Enter password:
```

下面在 db2 中进行数据恢复, 如下所示。

```
[root@localhost ~]# cd /usr/local/mysql/bin/
[root@localhost bin]# mysqldump -user=root -p test < /test.sql
Enter password:
```

步骤 06 值得注意的是, 在实验的过程中, 需要关闭系统自带的防火墙, 如下所示。

```
[root@localhost ~]# service iptables stop
```

步骤 07 db1 和 db2 服务器相互通告二进制日志的位置并启动复制功能。在 db1 服务上执行如下操作, 启动复制功能。

```
mysql> stop slave;
Query OK, 0 rows affected (0.01 sec)

mysql> change master to
-> master_host='192.168.0.3',
-> master_user='replication',
-> master_password='replication_password',
-> master_log_file='mysql-bin.000022',
-> master_log_pos=107;
Query OK, 0 rows affected (0.00 sec)

mysql> start slave;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> show variables like 'server%';
```

| Variable_name | Value |
|---------------|-------|
| server_id | 1 |

```
1 row in set (0.00 sec)
```

步骤 08 在 db2 服务上执行如下操作，启动复制功能。

```
mysql> stop slave;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> change master to
```

```
-> master_host='192.168.0.2',
```

```
-> master_user='replication',
```

```
-> master_password='replication_password',
```

```
-> master_log_file='mysql-bin.000022',
```

```
-> master_log_pos=107;
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> start slave;
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> show variables like 'server_id';
```

| Variable_name | Value |
|---------------|-------|
| server_id | 2 |

```
1 row in set (0.00 sec)
```

步骤 09 接下来，在 db1 中测试下看看是否 db1 和 db2 实现同步，如下所示。

```
mysql> show slave status\G;
```

```
***** 1. row *****
```

```
Slave_IO_State: Waiting for master to send event
```

```
Master_Host: 192.168.0.3
```

```
Master_User: replication
```

```
Master_Port: 3306
```

```
Connect_Retry: 60
```

```
Master_Log_File: mysql-bin.000025
```

```
Read_Master_Log_Pos: 107
```

```
Relay_Log_File: localhost-relay-bin.000005
```

```
Relay_Log_Pos: 253
```

```
Relay_Master_Log_File: mysql-bin.000025
```



```

Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB: test
.....
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 2
1 row in set (0.00 sec)

```

步骤 10 下面在 db2 中测试下看看是否 db1 和 db2 实现同步，如下所示。

```

mysql> show slave status\G;
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 192.168.0.2
Master_User: replication
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000023
Read_Master_Log_Pos: 107
Relay_Log_File: localhost-relay-bin.000003
Relay_Log_Pos: 253
Relay_Master_Log_File: mysql-bin.000023
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB: test
.....
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 1
1 row in set (0.00 sec)

```

步骤 11 下面通过对数据的操作检验 db1 和 db2 是否同步。首先在 db1 数据库中新增加一个表，并添加一条数据，如下所示。

```

mysql> use test
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| t3              |
+-----+
1 row in set (0.06 sec)

mysql> create table t(data int);
Query OK, 0 rows affected (0.08 sec)

```



```
mysql> insert into t values(1);
Query OK, 1 row affected (0.01 sec)
```

步骤 12 接下来，在 db2 服务器上查询 t 表是否添加成功，然后，也向 db2 中添加一条数据。

```
mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| t              |
| t3             |
+-----+
2 rows in set (0.03 sec)
```

```
mysql> select * from t;
+-----+
| data |
+-----+
| 1    |
+-----+
1 row in set (0.03 sec)
```

```
mysql> insert into t values(2);
Query OK, 1 row affected (0.02 sec)
```

步骤 13 接下来，回到 db1 中测试，刚才 db2 中添加的数据是否同步过来了，如下所示。

```
mysql> select * from t;
+-----+
| data |
+-----+
| 1    |
| 2    |
+-----+
2 rows in set (0.05 sec)
```

至此，主主同步配置完成，接下来开始安装 Lvs 和 Keepalived。

12.4.2 Lvs+Keepalived 的安装

需要在 db1 和 db2 两台服务器上都安装 LVS，可以通过地址 <http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.24.tar.gz> 进行下载，具体安装步骤如下。

下面开始解压并安装 ipvsadm-1.24.tar.gz，如下所示。

```
wget
http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.24.tar.gz
ln -s /usr/src/kernels/2.6.18-164.el5-i686/ /usr/src/linux
tar zxvf ipvsadm-1.24.tar.gz
cd ipvsadm-1.24
make && make install
```

接下来需要同时在 db1 和 db2 两台服务器上安装 Keepalived，可以通过下面的地址进行下载 <http://www.keepalived.org/software/keepalived-1.1.19.tar.gz>，具体的安装步骤如下。

```
wget http://www.keepalived.org/software/keepalived-1.1.19.tar.gz
cd keepalived-1.1.19
./configure --prefix=/usr/local/keepalived
make
make install

cp /usr/local/keepalived/sbin/keepalived /usr/sbin/
cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/
cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/
mkdir /etc/keepalived
```

12.4.3 Lvs+Keepalived 的配置

首先，需要对 Master 的 keepalived 进行配置，如下所示。

```
vim /etc/keepalived/keepalived.conf
global_defs {
    notification_email {
        zhangxy@test.com
    }
    notification_email_from jiankong@test.com
    smtp_server mail.test.com
    smtp_connect_timeout 30
    router_id LVS1
}
vrrp_sync_group test {
    group {
        loadbalance
    }
}
vrrp_instance loadbalance {
    state MASTER
    interface eth0
    lvs_sync_daemon_interface eth0
    virtual_router_id 51
```



```

priority 180
advert_int 1
authentication {
    auth_type PASS
    auth_pass 1111
}
virtual_ipaddress {
    10.1.1.176 dev eth0 label eth0:1
}
}
virtual_server 10.1.1.176 3306 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    persistence_timeout 20
    protocol TCP
    sorry_server 10.1.1.75 3306
    real_server 10.1.1.113 3306 {
        weight 3
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3306
        }
    }
}
}
}

```

接下来, 需要对 Backup 主机的 keepalived 进行配置, 如下所示。

```

vim /etc/keepalived/keepalived.conf
global_defs {
    notification_email {
        zhangxy@test.com
    }
    notification_email_from jiankong@test.com
    smtp_server mail.test.com
    smtp_connect_timeout 30
    router_id LVS1
}

vrrp_sync_group test {
    group {
        loadbalance
    }
}
}

```



```

vrp_instance loadbalance {
    state BACKUP
    interface eth0
    lvs_sync_daemon_inteface eth0
    virtual_router_id 51
    priority 150
    advert_int 1
authentication {
    auth_type PASS
    auth_pass 1111
}
virtual_ipaddress {
    10.1.1.176 dev eth0 label eth0:1
}
}
virtual_server 10.1.1.176 3306 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    persistence_timeout 20
    protocol TCP
    sorry_server 10.1.1.75 3306
    real_server 10.1.1.113 3306 {
        weight 3
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3306
        }
    }
}
}

```

接下来，需要对 Master 和 Backup 的 realserver 进行配置，对于 realserver 的配置 master 和 backup 是一致的，如下所示。

```

vim /etc/rc.d/init.d/realserver.sh
#!/bin/bash
# description: Config realserver lo and apply noarp
SNS_VIP=10.1.1.176
/etc/rc.d/init.d/functions
case "$1" in
start)
ifconfig lo:0 $SNS_VIP netmask 255.255.255.255 broadcast $SNS_VIP
/sbin/route add -host $SNS_VIP dev lo:0
echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore

```

```

echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
sysctl -p >/dev/null 2>&1
echo "RealServer Start OK"
;;
stop)
ifconfig lo:0 down
route del $SNS_VIP >/dev/null 2>&1
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce
echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
echo "RealServer Stopped"
;;
*)
echo "Usage: $0 {start|stop}"
exit 1
esac
exit 0

```

12.4.4 Master 和 Backup 的启动

首先, 需要先启动 Master 和 Backup 的 MySQL 服务器, 之后需要启动 keepalived 和 realserver 脚本, 如下所示。

```

/etc/rc.d/init.d/realserver.sh start
/etc/rc.d/init.d/keepalived start

```

将 keepalived 和 realserver 的启动脚本加入到 rc.local 自启动中:

```

echo "/etc/rc.d/init.d/realserver.sh start" >> /etc/rc.local
echo "/etc/rc.d/init.d/keepalived start" >> /etc/rc.local

```

12.5 MMM 高可用 MySQL 方案

MMM 即 MySQL 主主复制管理器 (Master-Master Replication Manager for MySQL) 是关于 MySQL 主主复制配置监控、故障转移和管理的一套可伸缩的脚本套件, 该套件也能对居于标准的主从配置的任意数量的从服务器进行读负载均衡, 可以实现数据备份、节点之间重新同步功能。

12.5.1 MMM 的架构

MMM 高可用 MySQL 架构方案中，如果当前的服务器挂掉后，会将后端的从服务器自动转向新的主服务器进行同步复制，不需要手工更改同步配置。该方案是目前比较成熟的解决方案。MySQL 本身没有提供 Replication Failover 的解决方案，通过 MMM 方案能实现服务器的故障转移，从而实现 MySQL 的高可用。下面先了解 MMM 高可用 MySQL 方案的架构图，如图 12-5 所示。

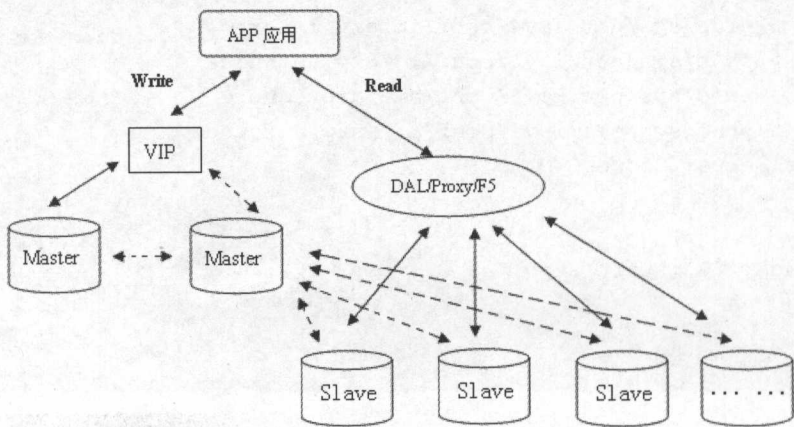


图 12-5 MMM 高可用 MySQL 方案的架构图

MMM 高可用 MySQL 架构解决了很多问题，特别是 read/write 比较高的 web 2.0 应用中。该架构存在如下的一些优点和缺点，先来分析下该方案的一些优点。

- 安全性、稳定性比较高，可扩展性好，高可用，当主服务器挂掉以后，另一个主服务器立即接管，其他的从服务器能自动切换，不需要人工干预。
- 写操作全部在主节点进行，并由从服务器数据库节点定时读取主服务器的二进制日志。
- 将众多客户的读请求分散到更多的数据库节点，从而减轻了单点的压力。

该方案也存在一些缺点，例如该方案中至少需要三个节点，对主机的数量有要求，需要实现读写分离，对程序来说难度系数较高。

MMM 高可用方案适合数据库访问量大，业务增长快，并且能够实现读写分离的场景，下面进行 MMM 高可用方案的实战。

12.5.2 配置环境

MMM 高可用方案是对 MySQL Master-Slave Replication 的一个补充，MMM 有三个重要的部分，包括 mmm_mon、mmm_agent 和 mmm_control。每一个 MySQL 服务器节点上都需要运行 mmm_agent，同时需要使一个机器运行 mmm_mon，mmm_mon 的作用主要用来监控其他数据库节点运行的状况，可以是独立的一台机器，也可以是和 AppServer 共享同一个服务器。

MMM 利用了 VIP（虚拟 IP）的技术，1 个网卡可以同时使用多个 IP，当某个数据库节点出现故障的时候，mmmd_mon 检查不到 mmmd_agent 对应 MySQL 服务器的状态，mmmd_mon 此时会下指令给某个正常的数据库节点的 mmmd_agent。

下面先准备下 MMM 高可用方案所需的实战环境，在如下的实战环境中，需要 4 台服务器，设置如表 12-4 所示。

表 12-4 MMM 高可用方案服务器列表

| 服务器 | IP 地址 | Server ID | 主机名称 | 操作系统 | MySQL |
|-----------------|-------------|-----------|------|----------|--------|
| Monitoring host | 192.168.0.5 | -- | mon | Fedora14 | 5.5.24 |
| Master 1 | 192.168.0.2 | 1 | db1 | Fedora14 | 5.5.24 |
| Master 2 | 192.168.0.3 | 2 | db2 | Fedora14 | 5.5.24 |
| Slave 1 | 192.168.0.4 | 3 | db3 | Fedora14 | 5.5.24 |

虚拟 IP 列表的角色和描述如表 12-5 所示。

表 12-5 MMM 高可用方案虚拟 IP 列表

| VIP | 角色 | 描述 |
|--------------|-------|------------------------------------------------|
| 192.168.0.12 | Read | 应用配置的读取 IP，也可以在前段加 LVS 等，做负载均衡。三台数据库每台一个浮动 VIP |
| 192.168.0.13 | Read | |
| 192.168.0.14 | Read | |
| 192.168.0.15 | Write | 应用配置的写入 VIP，单点写入 |

分别在各机器上安装 MySQL 服务，此时需要创建三个不同的用户，如表 12-6 所示。

表 12-6 创建不同用户的权限和功能

| 用户功能 | 描述 | 权限 |
|--------------|-----------------------------------|---------------------------------|
| Monitor User | 提供 Monitor Server 检查 MySQL 服务的状况 | Replication Client |
| Agent User | MMM agent 用来切换 replication master | Super,replicationClient,Process |
| Replication | 复制使用 | Replication Slave |

可以通过如下语句来创建三种不同功能的用户。

```
mysql> grant replication client on *.* to 'mmm_monitor'@'192.168.0.%' identified
by 'monitor_password';
Query OK, 0 rows affected (0.07 sec)

mysql> grant super,replication client,process on *.* to
```

```
'mmm_agent'@'192.168.0.%' identified by 'agent_password';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> grant replication slave on *.* to 'replication'@'192.168.0.%' identified
by 'replication_password';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

值得注意的是, mmm_monitor 用户使用在 Monitor Server(192.168.0.5)中, 另外 mmm_agent 用户和 replication 用户是在其他机器中使用的。

在进行 MMM 配置之前需要对 db1 和 db2 两台机器进行主主同步的相关配置, db1 和 db3 实现主从同步。主主同步配置(请参考本章的 12.3 小节)完成后, 接下来需要将 db1 和 db3 服务器设置成主从同步配置, db3 相关配置步骤如下。

步骤 01 首先, 在 db3 中进行数据恢复, 如下所示。

```
[root@localhost ~]# cd /usr/local/mysql/bin/
[root@localhost bin]# mysqldump -user=root -p test < /test.sql
Enter password:
```

步骤 02 接下来, 需要配置 my.cnf 文件, 如下所示。

```
server-id = 3
replicate-do-db = test
slave-skip-errors = all
sync_binlog = 1
```

步骤 03 设置主机的地址, 实现 db1 和 db3 主从复制, 如下所示。

```
mysql> stop slave;
Query OK, 0 rows affected (0.02 sec)

mysql> change master to
-> master_host='192.168.0.2',
-> master_user='replication',
-> master_password='replication_password';
-> master_log_file='mysql-bin.000022';
-> master_log_pos=107;
Query OK, 0 rows affected (0.04 sec)

mysql> start slave;
Query OK, 0 rows affected (0.00 sec)
```

步骤 04 在 db3 上面测试下看看是否 db1 和 db3 实现主从同步, 如下所示。

```
mysql> show slave status\G;
```

```

***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 192.168.0.2
Master_User: replication
Master_Port: 3306
Connect_Retry: 60
Master_Log_File: mysql-bin.000023
Read_Master_Log_Pos: 572
Relay_Log_File: localhost-relay-bin.000003
Relay_Log_Pos: 718
Relay_Master_Log_File: mysql-bin.000023
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB: test
.....
Last_SQL_Errno: 0
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 1
1 row in set (0.00 sec)

```

步骤05 在 db1 中新建一个表，然后添加一条数据，如下所示。

```

mysql> create table t2(data varchar(20));
Query OK, 0 rows affected (0.33 sec)

mysql> insert into t2 values('db1 insert a data');
Query OK, 1 row affected (0.08 sec)

```

步骤06 在从服务器上测试，查看该条记录是否同步成功，如下所示。

```

mysql> use test;
Database changed
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| t               |
| t2              |
| t3              |
+-----+
3 rows in set (0.00 sec)

mysql> select *from t2;
+-----+
| data          |
+-----+

```



```
| db1 insert a data |
+-----+
1 row in set (0.01 sec)
```

至此，db1 和 db2 完成主主同步，db1 和 db3 完成主从同步，MMM 高可用方案的准备工作已经完成，接下来小节开始对 MMM 进行安装和配置。

12.5.3 MMM 的安装

MMM 软件是基于 Perl 的。所以在安装的过程中需要安装许多 Perl 的模块，具体步骤如下所示。

步骤 01 下载并安装 MMM 软件，并在 db1、db2、db3 机器上配置 mysql-mmm-agent，mon 机器上配置 mysql-mmm-monitor。

下载地址：<http://mysql-mmm.org/downloads>
版本：mysql-mmm-2.2.1.tar.gz

步骤 02 下面在各个机器上开始安装，如下所示。

```
[root@localhost ~]# tar zxf mysql-mmm-2.2.1.tar.gz
[root@localhost ~]# cd mysql-mmm-2.2.1
[root@localhost mysql-mmm-2.2.1]# make install
```

步骤 03 安装许多 perl 的模块。可以根据官方的说明文档 <http://mysql-mmm.org/mmm2:guide>，严格按照版本号查找 rpm 包进行安装。下载的网址是 <http://pkgs.repoforge.org/>，下载需要的 rpm 包，然后进行安装即可。

12.5.4 Monitor 服务器的配置

MMM 的配置文件是在 /etc/mysql-mmm 目录下面，monitor 服务器需要配置该目录下面的 mmm_common.conf、mmm_mon.conf 两个文件。Mmm_common.conf 文件在 mmm 的各个节点都是一样的，因此配置完成之后可以直接复制到其他各个 DB 节点即可。具体操作步骤如下：

步骤 01 下面开始配置 mmm_monmm.conf 文件，如下所示。

```
[root@localhost ~]# vi /etc/mysql-mmm/mmm_common.conf
active_master_role  writer

<host default>
cluster_interface    eth2

pid_path              /var/run/mmm_agentd.pid
bin_path              /usr/lib/mysql-mmm/

replication_user      replication
```

```

    replication_password    replication_password

    agent_user              mmm_agent
    agent_password          agent_password
</host>

<host db1>
    ip                      192.168.0.2
    mode                    master
    peer                    db2
</host>

<host db2>
    ip                      192.168.0.3
    mode                    master
    peer                    db1
</host>

<host db3>
    ip                      192.168.0.4
    mode                    slave
</host>

<role writer>
    hosts                   db1, db2
    ips                     192.168.0.15
    mode                    exclusive
</role>

<role reader>
    hosts                   db1, db2, db3
    ips                     192.168.0.12, 192.168.0.13, 192.168.0.14
    mode                    balanced
</role>

```

步骤 02 把配置后的 mmm_conmm.conf 文件复制到其他各个节点上。

步骤 03 接下来，需要配置 mon 机器上的 mmm_mon.conf 配置文件，如下所示。

```

[root@localhost ~]# vi /etc/mysql-mmm/mmm_mon.conf
include mmm_common.conf

<monitor>
    ip                      127.0.0.1
    pid_path                /var/run/mmm_mond.pid
    bin_path                /usr/lib/mysql-mmm/

```

```

status_path          /var/lib/misc/mmm_mond.status
ping_ips             192.168.0.2, 192.168.0.3, 192.168.0.4
</monitor>

<host default>
monitor_user         mmm_monitor
monitor_password     monitor_password
</host>

debug 0

```

12.5.5 各个数据库服务器的配置

MMM 配置各个数据库服务器之前需要保持各个服务器上面的 mmm_comm.conf 文件内容一致。下面对每个服务器的 mmm_agent.conf 文件进行配置，具体步骤如下。

步骤 01 配置 db1 上面的 Agent 服务器的 mmm_agent.conf 配置文件，如下所示。

```

[root@localhost ~]# vi /etc/mysql-mmm/mmm_agent.conf
include mmm_common.conf
this db1

```

步骤 02 配置 db2 上面的 Agent 服务器的 mmm_agent.conf 配置文件，如下所示。

```

[root@localhost ~]# vi /etc/mysql-mmm/mmm_agent.conf
include mmm_common.conf
this db2

```

步骤 03 配置 db3 上面的 Agent 服务器的 mmm_agent.conf 配置文件，如下所示。

```

[root@localhost ~]# vi /etc/mysql-mmm/mmm_agent.conf
include mmm_common.conf
this db3

```

12.5.6 MMM 的管理

在启动 MMM 的时候，数据库机器需要启动 mmm-agent，命令如下。

```

[root@localhost ~]# /etc/init.d/mysql-mmm-agent start

```

Monitor 机器需要启动 mmm-monitor，命令如下。

```

[root@localhost ~]# /etc/init.d/mysql-mmm-monitor start

```

停止 MMM 只需要执行如下命令即可。

```

[root@localhost ~]# /etc/init.d/mysql-mmm-agent stop
[root@localhost ~]# /etc/init.d/mysql-mmm-monitor stop

```

此时在 mon 机器上，可以通过如下命令，查看服务器启动情况。


```
[root@localhost ~]# mmm_control show
```

如果需要将 db1 设置为 online 状态, 此时会配置 db1 的 VIP, 需要执行如下命令。

```
[root@localhost ~]# mmm_control set_online db1
```

在实现高可用方案中, 系统各个层面都需要被监控起来, 例如监控 Monitor 进程的状态, Agent 服务器进程的状态, MySQL 可用性的监控 数据库同步状态的监控 建议大家使用 nagios (请参考本书的相关内容) 进行监控。

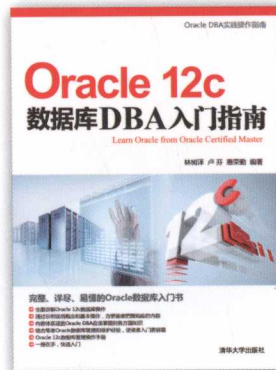
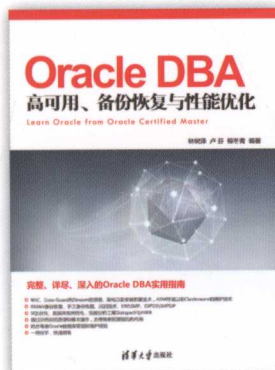
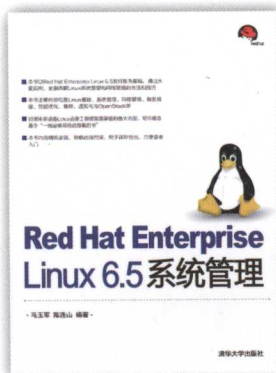
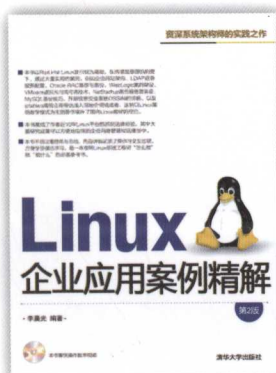
12.6 小结

前面的章节介绍的 MySQL Cluster 将是拥有最高可用性级别的解决方案, 同时解决了数据库写扩展的难题。本章介绍了各种搭建数据库冗余的高可用方案, 希望大家能在实验环境中实际的练习下, 知道如何配置是次要的, 关键是灵活地运用, 在生产环境中遇到问题如何解决, 如何进行日常的维护和监控, 安装配置只是一个开始。

MySQL 的高可用方案可供选择的方案相当多, 面对这么多的方案, 首先应该了解公司的业务, 了解应用系统中哪些会影响高可用, 以及通过对各个高可用方案对比不难找出适合自己公司业务的 MySQL 高可用方案。



图书推荐



清华大学出版社数字出版网站

WQBook 书文局泉

www.wqbook.com

ISBN 978-7-302-42043-9



9 787302 420439 >

定价：79.00元